

**Aladjev V.Z. and Bogdevicius M.A.**

**Maple: Programming,  
Physical and Engineering Problems**



*Fultus™ Books*



# Maple: Programming, Physical and Engineering Problems

by

Aladjev V.Z. and Bogdevicius M.A.

ISBN 1-59682-080-2

Copyright © 2006 by Aladjev V.Z. and Bogdevicius M.A.

All rights reserved.



Published by Fultus Publishing

Publisher Web Site: [www.fultus.com](http://www.fultus.com)

Fultus eLibrary: [elibrary.fultus.com](http://elibrary.fultus.com)

Online Book Superstore: [store.fultus.com](http://store.fultus.com)

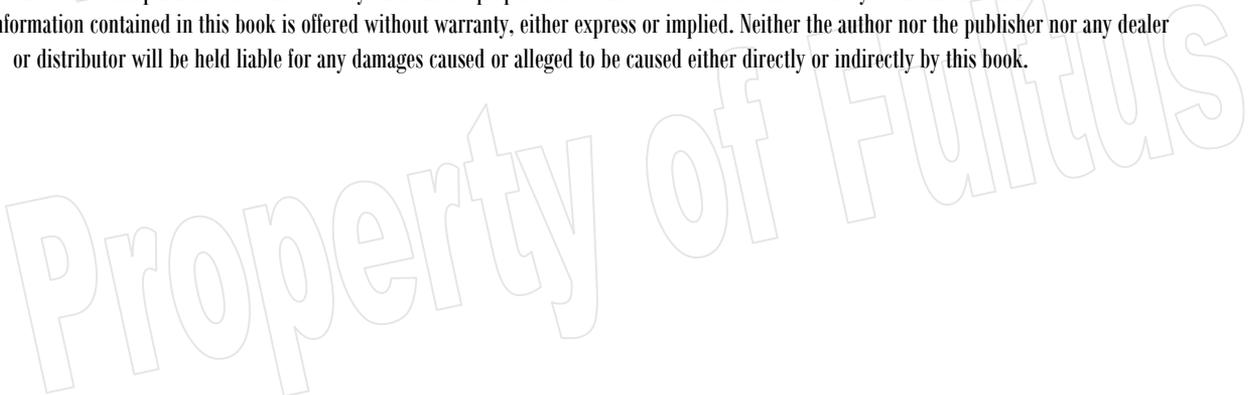
Writer web site: [writers.fultus.com/aladjev/](http://writers.fultus.com/aladjev/)



No part of this book may be used or reproduced in any manner whatsoever without written permission except in the case of brief quotations embodied in reviews and critical articles.

The author and publisher have made every effort in the preparation of this book to ensure the accuracy of the information.

However, the information contained in this book is offered without warranty, either express or implied. Neither the author nor the publisher nor any dealer or distributor will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.



# Table of Contents

Preface .....	11
Part1. New software for package <i>Maple</i> of releases 6 – 10.....	17
Chapter 1. General purpose software .....	19
Chapter 2. Software for work with procedural and modular <i>Maple</i> objects.....	33
Chapter 3. Software for work with symbols, strings, lists, sets and tables.....	49
Chapter 4. Software to support bit-by-bit processing of symbolic information.....	55
Chapter 5. Software that extends and improves the standard tools of <i>Maple</i> .....	62
Chapter 6. Software of working with <i>Maple</i> datafiles and documents.....	75
6.1. General purpose software .....	75
6.2. Software for operation with BINARY datafiles .....	82
6.3. Software for operation with <i>Maple</i> files.....	84
Chapter 7. Certain useful tools for work in <i>Maple</i> .....	87
Part2. Application of <i>Maple</i> for solution of engineering-physical problems .....	112
Chapter 8. The base problems of thermal conduction .....	113
8.1. Linear stationary problem of thermal conduction .....	113
8.1.1. Calculated equations of the linear stationary process of heat exchange.....	113
8.1.2. Input data for the solution of the problem.....	117
8.1.3. Brief description of Heat_st_linear program to solve the problem .....	119
8.1.4. An example of use of <i>Maple</i> -program Heat_st_linear .....	119
8.2. Nonlinear stationary problem of thermal conduction.....	122
8.2.1. Calculated equations of the nonlinear stationary process of heat exchange.....	122
8.2.2. Input data for the solution of the problem.....	124
8.2.3. Brief description of the Heat_st_nonlinear program solving the problem .....	126
8.2.4. An example of use of <i>Maple</i> -program Heat_st_nonlinear .....	127
8.3. Linear non-stationary problem of thermal conduction.....	129
8.3.1. Calculated equations of a linear non-stationary process of heat exchange.....	130
8.3.2. Input data for the solution of the problem.....	131
8.3.3. Brief description of Heat_nonst_linear program solving the problem.....	133
8.3.4. An example of use of <i>Maple</i> -program Heat_nonst_linear .....	134
8.4. Nonlinear non-stationary problem of thermal conduction .....	137
8.4.1. Calculated equations of nonlinear non-stationary process of heat exchange .....	137
8.4.2. Input data for the solution of the problem.....	138
8.4.3. Brief description of Heat_nonst_nonlinear program solving the problem.....	140
8.4.4. An example of the use of <i>Maple</i> -program Heat_nonst_nonlinear.....	140

Chapter 9. Basic problems of the elasticity theory.....	144
9.1. Definition of the geometrical characteristics of plane sections.....	144
9.1.1. Calculated expressions for definition of the geometrical characteristics of plane sections .....	144
9.1.2. Input data for the solution of the problem.....	147
9.1.3. Brief description of the Geometry program solving the problem.....	148
9.1.4. An example of use of the <i>Maple</i> -program Geometry .....	148
9.2. Computer-based calculation of the beam systems .....	149
9.2.1. Calculated equations of the beam systems .....	149
9.2.2. Input data for the solution of the problem.....	152
9.2.3. Brief description of the Strypas program solving the problem.....	154
9.2.4. An example of use of the <i>Maple</i> -program Strypas .....	154
9.3. Plane problem of the elasticity theory .....	156
9.3.1. The calculated equations of a plane problem of the elasticity theory.....	156
9.3.2. Input data for the solution of the problem.....	160
9.3.3. Brief description of the Plane program solving the problem .....	162
9.3.4. An example of use of the <i>Maple</i> -program Plane.....	162
9.4. Problem of contact of the elastic bodies .....	164
9.4.1. The calculated equations of the problem of contact of elastic bodies .....	164
9.4.2. Input data for the solution of the problem.....	167
9.4.3. Brief description of the Contact program solving the problem .....	169
9.4.4. An example of use of the <i>Maple</i> -program Contact .....	169
Chapter 10. Dynamic problems of the theory of elasticity.....	173
10.1. The dynamic problem of beam systems .....	173
10.1.1. The calculated equations of beam system's dynamics.....	173
10.1.2. Input data for the solution of the problem.....	176
10.1.3. Brief description of the Dynamic_strypas program solving the problem .....	178
10.1.4. An example of the use of <i>Maple</i> -program Dynamic_strypas .....	178
10.2. The plane problem of the theory of elasticity at dynamic loads .....	181
10.2.1. The calculated equations of the theory of elasticity problem at dynamic loads.....	182
10.2.2. Input data for the solution of the problem.....	183
10.2.3. Brief description of the Dynamic_plane program solving the problem.....	185
10.2.4. An example of the use of the <i>Maple</i> -program Dynamic_plane .....	186
10.3. Dynamic problem of shells of an arbitrary contour .....	190
10.3.1. The calculated equations of the shells' problem .....	190
10.3.2. Input data for the solution of the problem.....	194
10.3.3. Brief description of the Dynamic_shell program solving the problem .....	197
10.3.4. An example of use of the <i>Maple</i> -program Dynamic_shell .....	197
10.4. Geometrically nonlinear problem of the theory of elasticity at dynamic loadings .....	202
10.4.1. The calculated equations of the nonlinear problem .....	202
10.4.2. Input data for the solution of the problem.....	207
10.4.3. Brief description of the Dynamic_body program solving the problem.....	208

10.4.4. An example of use of the <i>Maple</i> -program <i>Dynamic_body</i> .....	209
<b>Chapter 11. Basic hydromechanics problems</b> .....	<b>210</b>
11.1. Nonvortical motion of a fluids, described by potential of velocities.....	210
11.1.1. The calculated equations for potential motion of a fluid .....	210
11.1.2. Input data for the solution of the problem.....	214
11.1.3. Brief description of the <i>Pot_flow</i> program solving the problem .....	215
11.1.4. An example of use of the <i>Maple</i> -program <i>Pot_flow</i> .....	216
11.2. Nonvortical motion of a fluid described by the stream function.....	221
11.2.1. The calculated equations for motion of a fluid as the stream function .....	221
11.2.2. Input data for the solution of the problem.....	223
11.2.3. Brief description of the <i>Stream_flow</i> program solving the problem.....	225
11.2.4. An example of use of the <i>Maple</i> -program <i>Stream_flow</i> .....	225
11.3. The Navier-Stokes's equations represented in the terms of velocity-pressure .....	229
11.3.1. The calculated equations of Navier-Stokes.....	229
11.3.2. Input data for solution of the problem .....	232
11.3.3. Brief description of the <i>Navier_Stokes</i> program solving the problem .....	233
11.3.4. An example of use of the <i>Maple</i> -program <i>Navier_Stokes</i> .....	234
11.4. Solution of the Navier-Stokes's equations, written in the terms of variables of the vortex and stream function .....	237
11.4.1. The calculated equations of Navier-Stokes in variables of vortex and stream function .....	237
11.4.2. Input data for the solution of the problem.....	239
11.4.3. Brief description of the <i>Vort_Stream</i> program solving the problem .....	242
11.4.4. An example of use of the <i>Maple</i> -program <i>Vort_Stream</i> .....	242
11.5. Non-stationary motion of compressible fluid, described by potential of velocity.....	245
11.5.1. The calculated equations of non-stationary potential motion of a fluid .....	245
11.5.2. Input data for the solution of the problem.....	247
11.5.3. Brief description of the <i>Dpot_flow</i> program solving the problem.....	249
11.5.4. An example of use of the <i>Maple</i> -program <i>Dpot_flow</i> .....	249
<b>Chapter 12. Some applied problems of hydromechanics</b> .....	<b>252</b>
12.1. Equation of Reynolds for a layer of lubrication .....	252
12.1.1. The calculated equations of Reynolds .....	252
12.1.2. Input data for the solution of the problem.....	255
12.1.3. Brief description of the <i>Reynold</i> program solving the problem .....	256
12.1.4. An example of use of the <i>Maple</i> -program <i>Reynold</i> .....	256
12.2. A model of motion of a non-Newtonian fluid .....	259
12.2.1. The calculated equations of motion of a non-Newtonian fluid.....	260
12.2.2. Input data for the solution of the problem.....	263
12.2.3. Brief description of the <i>NonNiuton</i> program solving the problem.....	265
12.2.4. An example of use of the <i>Maple</i> -program <i>NonNiuton</i> .....	265
12.3. A model of convective heat exchange in a fluid.....	268
12.3.1. The calculated equations of nonisothermal convective heat exchange .....	268

---

12.3.2. Input data for the solution of the problem.....	271
12.3.3. Brief description of the Heat_flow program solving the problem .....	274
12.3.4. An example of use of the Maple-program Heat_flow .....	274
12.4. Problem of heat exchange and mass transfer for an incompressible fluid .....	278
12.4.1. The calculated equations of heat exchange and mass transfer .....	278
12.4.2. Input data for the solution of the problem.....	280
12.4.3. Brief description of the Heat_mass_flow program solving the problem .....	284
12.4.4. An example of use of the Maple-program Heat_mass_flow.....	284
12.5. Models of motion of a fluid in a hydraulic system .....	288
12.5.1. Motion of a compressible fluid in a hydraulic system.....	288
12.5.2. Input data for the solution of the problem.....	294
12.5.3. Brief description of the Hydro program solving the problem .....	295
12.5.4. An example of use of the Maple-program Hydro .....	296
<b>Chapter 13. Applied problems of mechanics - 1.....</b>	<b>298</b>
13.1. Calculation of moments of inertia of a solid body .....	298
13.1.1. Calculated expressions for definition of moments of inertia of a solid body .....	298
13.1.2. Input data for the solution of the problem.....	300
13.1.3. Brief description of the Mass_inertia program solving the problem .....	301
13.1.4. An example of use of the Maple-program Mass_inertia .....	301
13.2. Calculation of inertia moments of a system of solid bodies.....	303
13.2.1. The calculated expressions for determination of inertia moments .....	304
13.2.2. Input data for the solution of the problem.....	305
13.2.3. Brief description of the Mass_system program solving the problem.....	306
13.2.4. An example of use of the Maple-program Mass_system .....	306
13.3. Nonlinear oscillations of mechanical systems .....	308
13.3.1. The calculated expressions for description of nonlinear oscillations.....	308
13.3.2. Input data for the solution of the problem.....	310
13.3.3. Brief description of the Amplitude program solving the problem .....	311
13.3.4. An example of use of the Maple-program Amplitude.....	311
13.4. Derivation and solution of equations of motion of a mechanical system with the concentrated parameters.....	313
13.4.1. The calculated expressions for deduction of equations of motion of a mechanical system.....	313
13.4.2. Input data for the solution of the problem.....	317
13.4.3. Brief description of the Pendium program solving the problem .....	317
13.4.4. An example of use of the Maple-program Pendium .....	317
13.5. Calculation of transition torsion oscillations in a mechanical transmission .....	320
13.5.1. The calculated expressions for determination of torsional oscillations .....	320
13.5.2. Input data for the solution of the problem.....	323
13.5.3. Brief description of the Driver program solving the problem .....	325
13.5.4. An example of use of the Maple-program Driver .....	325

<b>Chapter 14. The applied problems of mechanics - 2</b> .....	<b>328</b>
14.1. Motion of a vehicle on a rough road.....	328
14.1.1. The calculated expressions for definition of motion of a vehicle.....	328
14.1.2. Input data for the solution of the problem.....	331
14.1.3. Brief description of the Trailer program solving the problem.....	332
14.1.4. An example of use of the <i>Maple</i> -program Trailer.....	332
14.2. Stationary random oscillations of a vehicle.....	335
14.2.1. The calculated expressions for determination of motion of a vehicle.....	335
14.2.2. Input data for the solution of the problem.....	337
14.2.3. Brief description of the Trailer_random program solving the problem.....	338
14.2.4. An example of use of the <i>Maple</i> -program Trailer_random.....	339
14.3. Motion of a carriage along a rough railway.....	340
14.3.1. The calculated expressions for determination of a carriage motion.....	340
14.3.2. Input data for the solution of the problem.....	342
14.3.3. Brief description of the Carriage program solving the problem.....	343
14.3.4. An example of use of the <i>Maple</i> -program Carriage.....	344
14.4. Dynamics of pneumatic vibration-extinguisher with stationary magnets.....	345
14.4.1. Equation of a vibration-extinguisher motion with stationary magnets.....	346
14.4.2. Input data for the solution of the problem.....	348
14.4.3. Brief description of the Damper program solving the problem.....	348
14.4.4. An example of use of the <i>Maple</i> -program Damper.....	349
14.5. Models of gas motion in a pneumatic system.....	351
14.5.1. Motion of gas in a pneumatic system.....	352
14.5.2. Input data for the solution of the problem.....	357
14.5.3. Brief description of the Pneumo program solving the problem.....	359
14.5.4. An example of use of the <i>Maple</i> -program Pneumo.....	359
<b>Chapter 15. Application of <i>Maple 6</i> for solution of optimization problems</b> .....	<b>361</b>
15.1. Search methods for unconstrained optimization.....	361
15.1.1. Nelder and Mead's method.....	362
15.1.1.1. Input data for the solution of the problem.....	364
15.1.1.2. Brief description of the Nelder_Mead program solving the problem.....	364
15.1.1.3. An example of use of the <i>Maple</i> -program Nelder_Mead.....	364
15.1.2. Hooke and Jeeves' method.....	364
15.1.2.1. Input data for the solution of the problem.....	365
15.1.2.2. Brief description of the Hooke_Jeeves program solving the optimization problem.....	366
15.1.2.3. An example of use of the <i>Maple</i> -program Hooke_Jeeves.....	366
15.1.3. Davidon's cubic interpolation method.....	366
15.1.3.1. Input data for the solution of the problem.....	368
15.1.3.2. Brief description of the Cubic_interpolation solving the optimization problem.....	369
15.1.3.3. An example of use of the <i>Maple</i> -program Cubic_interpolation.....	369
15.2. Gradient methods for unconstrained optimization.....	369

15.2.1. Gradient method .....	369
15.2.1.1. Input data for the solution of the problem .....	370
15.2.1.2. Brief description of the Gradient program solving the problem.....	370
15.2.1.3. An example of use of the <i>Maple</i> -program Gradient .....	370
15.2.2. Fletcher-Reeves and Polak-Ribiere method .....	371
15.2.2.1. Input data for the solution of the problem .....	372
15.2.2.2. Brief description of the Fletcher_Reeves_Polak_Ribiere program solving the problem.....	372
15.2.2.3. An example of use of the <i>Maple</i> -program Fletcher-Reeves-Polak-Ribiere .....	372
15.2.3. Newton’s methods .....	373
15.2.3.1. Input data for the solution of the problem .....	374
15.2.3.2. Brief description of the Newtons_method program solving the optimization problem.....	375
15.2.3.3. An example of use of the <i>Maple</i> -program Newtons_method.....	375
15.3. Quasi-Newton methods.....	375
15.3.1. Simple rank procedure .....	375
15.3.1.1. Input data for the solution of the problem .....	377
15.3.1.2. Brief description of the QUASI_SIMPLE program solving the optimization problem .....	377
15.3.1.3. An example of use of the <i>Maple</i> -program QUASI_SIMPLE .....	377
15.3.2. Davidon-Fletcher-Powell method .....	378
15.3.2.1. Algorithm of Davidon-Fletcher-Powell method .....	378
15.3.2.2. Input data for the solution of the problem .....	378
15.3.2.3. Brief description of the Davidon_Fletcher_Powell program solving the problem.....	379
15.3.2.4. An example of use of the <i>Maple</i> -program Davidon_Fletcher_Powell .....	379
15.3.3. Broyden-Fletcher-Goldfarb-Shanno method .....	379
15.3.3.1. Algorithm of Broyden-Fletcher-Goldfarb-Shanno method .....	379
15.3.3.2. Input data for the solution of the problem .....	380
15.3.3.3. Brief description of the BFGS program solving the optimization problem .....	380
15.3.3.4. An example of use of the <i>Maple</i> -program BFGS .....	380
15.3.4. Memoryless quasi-Newton method .....	380
15.3.4.1. Algorithm of memoryless quasi-Newton method .....	380
15.3.4.2. Input data for the solution of the problem .....	381
15.3.4.3. Brief description of the Memoryless_QN program solving the optimization problem .....	381
15.3.4.4. An example of use of the <i>Maple</i> -program Memoryless_QN .....	381
15.4. Constrained minimization methods .....	382
15.4.1. Penalty-function method .....	382
15.4.1.1. Fiacco and McCormick method .....	382
15.4.1.2. Input data for the solution of the problem .....	383
15.4.1.3. Brief description of the Fiacco_McCorm. program solving the optimization problem.....	384
15.4.1.4. An example of use of the <i>Maple</i> -program Fiacco_McCormick .....	384
15.4.2. Complex method .....	384
15.4.2.1. Algorithm of Complex method .....	385
15.4.2.2. Input data for the solution of the problem .....	385

15.4.2.3. Brief description of the Complex program solving the optimization problem .....	385
15.4.2.4. An example of use of the <i>Maple</i> -program Complex .....	386
15.4.3. Methods based on the sequence of linear programs. The first method .....	386
15.4.3.1. Cutting plane method.....	387
15.4.3.2. Input data for the solution of the problem .....	387
15.4.3.3. Brief description of the Cutting_Plane program solving the optimization problem .....	388
15.4.3.4. An example of use of the <i>Maple</i> -program Cutting_Plane.....	388
15.4.4. The method based on the sequence of linear programs. The second method.....	389
15.4.4.1. Method of approximate programming.....	389
15.4.4.2. Input data for the solution of the problem .....	389
15.4.4.3. Brief description of the SLP program solving the optimization problem .....	390
15.4.4.4. An example of use of the <i>Maple</i> -program SLP .....	390
15.5. Genetic algorithms .....	390
15.5.1. The procedure of genetic algorithm.....	392
15.5.2. Input data for the solution of the problem.....	392
15.5.3. Brief description of the Genetic_dalia program solving the optimization problem .....	393
15.5.4. An example of use of the <i>Maple</i> -program Genetic_dalia .....	393
References.....	394
The information about the writers.....	402

DISTRIBUTION

Property of Fultus



# Preface

*Computer Algebra* (also known as *Symbolic Computation* or *Computational Algebra*) has found application in many fields of science such as mathematics, physics, chemistry, computer science, engineering, education, technology, computational biology, etc. The *computer algebra systems (CAS)* such as *Maple*, *Reduce*, *MuPAD*, *Axiom*, *Macsyma*, *Mathematica*, *Derive*, *Magma*, and others are becoming more and more popular in teaching, research and industry. The area of symbolic and algebraic computation aims at automation of mathematical computations of all sorts. The resulting computer systems, both experimental and commercial, are powerful tools for scientists, engineers, and educators. This research combines mathematics with advanced computing techniques. *CAS* is an interdisciplinary area between *Mathematics* and *Computer Science*. Its research focuses on the development of algorithms for performing *symbolic* manipulations with algebraic objects on computers, and design of programming languages and environments for implementing these algorithms.

In a series of our books and papers [28-30, 32, 33, 41, 111-116, 120-122, 124-127, 235-247], such packages as *Maple*, *Reduce*, *MathCAD* and *Mathematica* have been considered. Our experience in the detailed testing and practical use of different mathematical and physical applications of four mathematical packages (*Reduce*, *Maple*, *MathCAD* and *Mathematica*) enables us to consider the packages *Maple* and *Mathematica* as undoubted leaders (*on the basis of a generalized index*) among all listed modern tools of computer algebra. Meanwhile, we give preference to the package *Maple* due to a number of strong reasons described in the above books enough in detail.

*Computer algebra* becomes a rather powerful and useful tool for scientists and experts from various fields. However, manifold applications demand that the essential significant numerical calculations be combined with algebraic ones. With each new release, the package *Maple* meets more and more requirements. The *Maple* package has been widely used not only as a tool of solving mathematical problems. The package allows us to revise approaches to teaching subjects related to mathematics in universities by defining in many cases the methods for a teaching of subjects with the use of PCs to solve mathematical problems for various purposes [111-116, 120-122, 124-127, 235-247].

Researchers use well-known *Maple* package as an essential tool when solving problems related to their investigation. The package is ideal for formulating, solving, and exploring different mathematical models. Its symbolic manipulation facilities extend greatly over a range of problems that can be solved with its help. Educators in high schools, colleges, and universities have revitalized traditional curricula by introducing problems and exercises, which use *Maple's* interactive mathematics and physics. Students can concentrate on the more fundamental concepts rather than on tedious algebraic manipulations. Finally, engineers and experts in industries use *Maple* as an efficient tool replacing many traditional resources such as reference books, calculators, spreadsheets and programming languages. These users easily solve mathematical problems, creating projects and consolidating their computations into professional reports.

*Maple products* embody advanced technologies such as symbolic computation, infinite precision numeric, innovative *Web* components, extensible user-interface technology, and an unrivalled suite of mathematical algorithms for intelligent management of complex mathematics. Over 3 million users benefit from advanced *Maple* technology. Virtually, all major universities and research

institutes in the world, including such as MIT, Oxford, Stanford and Waterloo, has adopted *Maple* products to enhance their education and research activities. Waterloo *Maple's* industrial customer base includes Boeing, Bosch, Canon, NASA, etc.

Meanwhile, our operational experience in the period of 1997 – 2005 with *Maple* of releases 4, 5, 6, 7, 8, 9/9.5 and 10 enabled us not only to estimate its advantages compared with other similar mathematical packages, but has also revealed a number of faults and shortcomings which were eliminated by us. Furthermore, *Maple* does not support a number of important procedures of information processing, *symbolic* and *numeric* computing, including the tools of access to datafiles. By operating *Maple* we have developed rather effective tools (*procedures* and *program modules*), largely extending the possibilities of the package. This software has been organized as a *Library* that is structurally similar to the main *Maple* library, and is provided with a rather detailed Help system analogous to *Maple* Help system.

In particular, *Maple* does not provide sufficient compatibility of releases 6-10. This fact and the incompatibility of the package found out by us at a level of base platforms – *Windows* 98SE and lower, on the one hand, and *Windows* XP and above, on the other hand, the decision of a compatibility problem for the *Library* means have demanded.

The tools represented in the *Library* increase the range and efficiency of use of the package on the *Windows* platform owing to the innovations in three basic directions: (1) *elimination of a series of basic defects and shortcomings*, (2) *extension of capabilities of a series of standard tools*, and (3) *replenishment of the package by new means which increase the capabilities of its program environment, including the facilities improving the compatibility of releases 6, 7, 8, 9/9.5 and 10*. The basic attention is devoted to additional tools created in the process of practical use and testing of the package of releases 4-8 which by some parameters considerably extend the capabilities of the package making the work with it much easier; a considerable attention is also devoted to the tools providing package compatibility of releases 6, 7, 8, 9/9.5 and 10. The experience in using the above software for various applications has confirmed its valuable operational characteristics.

It should be noted that a series of our books and papers on *Maple* problems [111-116, 120-122, 124-127, 235-247], representing tools developed by the authors and containing suggestions on further development of the package encouraged the development of such applications as package modules **FileTools**, **LibraryTools**, **ListTools** and **StringTools**. However, the means suggested by us essentially extend the capabilities of the package, which in many cases exceed those of the specified modules.

The above software has been organized into the user *Library*, whose the current version contains tools (*more than 640 procedures and program modules*) which are oriented to a wide area of computing and information processing. The *Library* is structurally similar to the main *Maple* library and is supplied with the advanced Help system about the tools located in it. In addition, it is logically connected with the main *Maple* library, providing access to the tools contained in it similarly to the package tools. The simple guide describes the installation of the *Library* at presence in the PC with the above-mentioned *Windows* platform of the installed *Maple* package of releases 6, 7, 8, 9/9.5 or 10.

Taking into account our long-term experience in operation with the *Maple* package of releases 4-10 and experience of our colleagues from universities and the academic institutes of Lithuania, Latvia, Belarus, Estonia and Russia, it is should be noted, that many of tools (*or their analogues*) of our *Library* are worth to be included into standard deliveries of subsequent *Maple* releases. At present, they are accessible to the *Maple* users as the offered *Library* supporting releases 6-10 and functioning on platforms *Windows* 95 and later. *Library* tools in many cases allow us to facilitate programming of various applied problems in the *Maple* environment of releases 6-10.

It is possible to state, that a series of our books on the *Maple* problems [120-122, 124-127, 235-247] that represent the means developed by us and contain useful tips on the further development of the package, has encouraged the development of package modules **FileTools**, **LibraryTools**, **ListTools** and **StringTools**. However, in this respect tools represented by us essentially extend capabilities of the package, exceeding those of the specified package modules in many cases.

The *Library* is designed for a wide audience of experts, teachers, post-graduates and students of natural-science professions who use *Maple* of releases 6-10 on *Windows* platform in their own professional work. The *Library* contains well-designed software (*a set of procedures and program modules*), which supplements well the already available *Maple* software with the orientation towards the widest circle of the *Maple* users, greatly enhancing its usability and effectiveness. Our experience reveals that the use of *Library* provides more opportunities of *Maple* of releases 6-10, simplifying the programming of various practical problems in its environment. This *Library* will be of special interest above all to those *who* use *Maple* not only as a highly intellectual calculator but also as environment for programming of different problems in their professional activities. The *Library* has been rewarded by "Smart Award" from Smart DownLoads Network.

Furthermore, the presence in the *Library* delivery of the text datafile "ProcUser.txt" with source codes of the *Library* software and *mws*-files with help-pages composing Help database of the *Library* allows us to adapt it to other underlying platforms different from *Windows*. Furthermore, the source codes, using both the effective and the non-standard technique, can serve as an useful enough practical programming guide on the *Maple* language. Archive with this *Library* and accompanying materials you may free-of-charge download from the web-site

<http://writers.fultus.com/aladjev/source/UserLib6789.zip>

Briefly, we shall present the contents of separate chapters of the book. The book consists of two parts: (1) new software for package *Maple* of releases 6-10 and (2) application of *Maple* for solution of physical & engineering problems. In the *first* chapter, the software providing the most general procedures of operation with package *Maple* of releases 6-10 is represented. Here and further every procedure is being represented in the following aspects: (1) *procedure's name with its brief characteristic*, (2) *procedure call sequence*, (3) *formal procedure parameters*, (4) *description of the procedure*, (5) *procedure source code*, and (6) *most typical examples of its use*. The source codes allow not only easily to immerse them into the *Maple* environment of releases 6-10 on many computer platforms, but also to use their as an useful enough illustrative material at a mastering of the advanced programming in the package environment. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

In the *second* chapter of the book, software for work with procedural and modular *Maple* objects are considered. These tools support kinds of processing such as: converting of modules into procedures; testing of presence in files of incorrect modules; check of parameters of procedures and modules; check of activity (*availability for direct use*) of a procedure or a module; check of type of a modular table; converting of files of the input *Maple* format containing modules; converting of a module of the second type into the first; converting of a file of the input *Maple* format into a file of the internal *Maple* format, and vice versa, etc. The represented tools provide many of the manifold useful operations with procedural and modular objects of *Maple*. The means represented in the given chapter allows to effectively program the above objects at solving your *Maple* appendices.

In the *third* chapter, software for work with symbols, strings, lists, sets and tables are considered. The means represented in the given chapter allows to effectively program your *Maple* appendices using these important *Maple* data structures. These tools provide a number of useful procedures such as special kinds of converting; comparison of strings or/and symbols; case sensitive pattern

searching; exhaustive substitutions into strings or symbols; inversion of symbols, strings or lists; reducing of multiplicity of entries of a symbol into a string; identification of entries of special symbols into a string; and others. The list structures play an extremely important role, defining the ordered sequences of elements. A series of procedures of the section supports useful kinds of processing such as: a special converting of lists into sets, and vice versa; operation with rarefied lists; dynamic assignment of values to elements of a list or a set; evaluation of indices of a table over its entry; representation of a special type of tables; special kinds of exhaustive substitutions into lists or sets; a series of important kinds of sorting of nested lists, and also many others. In a series of cases, these tools simplify operation with *Maple* objects of type  $\{string, symbol, list, set, table\}$  in the *Maple* environment. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

In the *fourth* chapter, software to support bit-by-bit processing of symbolic information are considered. The package does not possess tools of the similar type. The software offered by us is represented by six useful procedures such as *Bit*, *Bit1*, *xbyte*, *xbyte1*, *xNB* and *xpack*. These procedures serve for *bit-by-bit* information processing, i.e. the user has possibility to operate with strings or symbols at a level of separate bits composing them.

In the *fifth* chapter, the tools that extend and improve the standard *Maple* tools for releases 6-10 are represented. These tools are used enough widely both at operation with the *Maple* package in interactive mode and at programming of various problems in its environment. They represent undoubted interest at programming of various problems in the *Maple* environment, both by simplifying the programming and by making it by more clear.

The *sixth* chapter, software for work with *Maple* datafiles and documents are considered. Being the programming language in the package environment, oriented, first, to symbolic calculations (*computer algebra*) the *Maple* language has the relatively limited opportunities at operation with data that are located in external computer memory. Moreover, in this respect the *Maple* language essentially yields to traditional programming languages such as C, COBOL, FORTRAN, PL/1, Pascal, ADA, Basic, etc. At the same time, the *Maple* language, oriented, first of all, onto solution of problems of mathematical character, gives a set of tools for access to datafiles which can quite satisfy a broad enough audience of users of physical and mathematical appendices of the package. In the present chapter, additional means of access to datafiles are represented, essentially extending opportunities of the package in the given direction. Many of them simplify the programming of many problems dealing with access to datafiles of different purpose. With all evidence we can assert, that new package modules **FileTools** and **LibraryTools** have been inspired by a series of our works [124-147, 235-247] with which the *Maple* developers have been acquainted. However, our set of the similar procedures is considerably more representative and is focused on wider practical use.

The *seventh* chapter represents certain useful tools for work in *Maple*. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

From the *eighth* chapter, the book represents applied aspects of package *Maple* by an example of solution of a wide circle of engineering-physical problems from the following fields: classical mechanics, hydrodynamics, hydromechanics, heating engineering, etc. Each problem is represented in the following context: (1) a common theoretical part (*setting of a problem*), (2) initial data for solution of the problem, (3) brief description of *Maple*-program solving the problem, and (4) examples of application of the given program for the solution of concrete application with interpretation of obtained outcomes. Given that, source codes of the debugged *Maple*-programs with the data for test examples, which are appropriated to them, are represented in the archive attached to the present book. Similar organization of the material allows the reader to apply software immediately, which considered in the book, in the own professional activity. Archive with

these codes, test examples, test data and other accompanying materials you may free-of-charge download from the web-site

<http://writers.fultus.com/aladjev/source/Archive.zip>

The presented programs can serve as a final software product and as good material illustrating main functionalities of the package *Maple*. The material of the second part of the book is based on our previous books [124,125] and the outcomes of adapting engineering-physical problems are considered relative to releases 6-10 of *Maple*. Let us study the contents of the last seven chapters of the book, which compose its second part of the applied assignment.

The *eighth* chapter of the book considers questions of computer solution in the environment of package *Maple* of the basic problems of thermal conductivity having important value to solve many applied problems of thermal physics as well as a combination of problems in the theory of elasticity and plasticity. Solution in the environment of the package of the following problems is considered: linear (1) and nonlinear (2) stationary problems of thermal conductivity as well as linear (3) and nonlinear (4) non-stationary problems of thermal conductivity. In particular, linear non-stationary heat conductivity equation is one of the main equations describing convective heat exchange and mass exchange occurring in systems of various physical nature. Whereas the last problem has not only major independent value for the definition of non-stationary temperature fields in various matters and materials, but is an important component of more complex engineering-physical problems which have rather wide engineering-physical applications as well.

In the *ninth* chapter, four problems of linear mechanics of deformable bodies are considered. The main relations concerning the description of tensely-deformable state and the relations for resilient bodies are given. Each problem is solved by the finite element method. The main principles of solution of the next practically important problems are represented: (1) definition of geometrical parameters of the cross-sections of bodies, (2) calculation of rod constructions at static loads, (3) the plane problem of the theory of elasticity, and (4) a contact problem of two elastic bodies. The indicated problems represent an applied interest; therefore *Maple*-programs corresponding to them on a rather wide circle of the applications are oriented.

In the *tenth* chapter, the dynamic problems of the theory of elasticity representing major practical interest at designing many objects of mechanical engineering are considered. The main problems here are definition of own and forced oscillations of elastic bodies, and also research of behaviour of elastic systems at short-term loads. The problems, considered in the chapter, represent practical interest for research of the indicated questions. The main principles of solution of the next practically important problems are represented: (1) research of own and forced oscillations of linear rod systems, (2) the plane problem of the theory of elasticity at dynamical loads, (3) a dynamical problem of shells of arbitrary configuration, and (4) a geometrically nonlinear problem of the theory of elasticity at dynamical loads. The indicated problems represent an applied interest; therefore, *Maple*-programs corresponding to them on a rather wide circle of the applications are oriented.

In the *eleventh* chapter, five problems of hydrodynamics having the important applied value and allowing to solve many engineering problems, which are linked with the flow around bodies by a liquid, oscillations of constructions in a liquid, a percussion impulse action onto a liquid, etc. are considered. The following problems are represented: an vortex-free motion of a liquid in terms of potential of speeds (1) and the flow function (2); (3) the equations of Navier-Stokes describing common movement of a liquid; (4) a solution of Navier-Stokes equations in the terms of a vortical function of flow, widely used in applied problems of hydromechanics, and (5) a non-stationary movement of a compressed liquid, described by the potential of speeds.

In the *twelfth* chapter, some the following problems of hydromechanics, which play an important role in research of problems are considered: (1) hydrodynamic lubrication between movable surfaces, (2) models of movement of a non-Newtonian liquid, (3) the problem of convective heat exchange and mass transfer for an incompressible liquid, (4) a model of convective heat exchange in a liquid, and (5) the models of movement of a liquid in a hydraulic system. All these problems play an important role in many technical applications, above all, of hydro-mechanical character. The finite element method adapted for the environment of *Maple*-language of the package appears as the main tool of the solution of engineering-physical problems is represented in the book.

When solving dynamic problems of solid mechanics, there is a necessity of definition of moments of inertia of masses of bodies of complex geometrical forms as well as systems of bodies, with respect to various axes of coordinates. For example, the *thirteenth* chapter considers the problems on definition of moments of inertia of masses of a body with a complex geometrical form and of systems of such bodies. By the facilities of package *Maple*, the following interesting problems are studied: (1) calculation of moments of inertia of a solid; (2) a calculation of moments of inertia of the system of solids; (3) the nonlinear oscillations of mechanical systems; (4) derivation and solution of equations of motion of a mechanical system with concentrated parameters, and (5) calculation of transitional torsional oscillations in mechanical transmission.

The problems of moving of transport along uneven way are of great interest. When transport move along uneven way upon elements of its hanger bracket a rather large loads affect, which in turn, decrease a longevity of elements of hanger bracket and all carrier as a whole, and also essentially influence onto its comfortable quality. In this connection, the *fourteenth* chapter of the book considers the following interesting problems: (1) movements of a carrier on an uneven way; (2) the stationary random oscillations of a carrier; (3) movements of a car on an uneven railway path; (4) dynamics of pneumatic vibration extinguisher with steady magnets, and (5) the models of moving gas in the pneumatic system.

At last, the *fifteenth* chapter is devoted to application of *Maple* for solution of certain optimization problems. In this chapter, the problem of minimizing function of real variables under the assumption that no constraint is imposed on the values of these variables is considered in the light of different standpoints.

This book represents the extended presentation of materials of lectures on the fundamentals of work in the environment of mathematical package *Maple*, which were given by the authors in 2001-2004 for the researchers of the Baltic Branch of the International Academy of Noosphere (Tallinn, Estonia), Vilnius Gediminas Technical University (Vilnius, Lithuania), Grodno State University and Grodno branch of Institute of Modern Knowledge (Grodno, Byelorussia). The given courses aimed at the introduction of innovations of the new software for *Maple* with orientation, above all, on the experts in the field of physical and mathematical sciences and also of other priority naturally-scientific directions of modern academic researches and teaching of computer science and mathematical disciplines in universities and colleges.

In view of mathematical orientation of package *Maple* and widely used in this connexion of mathematical terminology and notation, at presentation of all next material we suppose the presence of a sufficient mathematical culture for the reader. Therefore, we will focus on the facilities and possibilities of the especially considered package, instead of purely mathematical subjects.

# Part 1.

New software

for package *Maple* of releases 6 – 10



# Chapter 1.

## General purpose software

In the given chapter, the software providing the most general procedures of operation with package *Maple* of releases 6-10 is represented. Here and further every procedure is being represented in the following aspects: (1) *procedure's name with its brief characteristic*, (2) *procedure call sequence*, (3) *formal procedure parameters*, (4) *description of the procedure*, (5) *procedure source code*, and (6) *most typical examples of its use*. The source codes allow not only easily to immerse them into the *Maple* environment of releases 6-10 on many computer platforms, but also to use their as an useful enough illustrative material at a mastering of the advanced programming in the package environment. Many of the software, represented here, use useful enough technique and methods in the practical respect and receptions of programming (including and *non-standard* ones) in the package environment, conditioning to them as the applied, and educational interest. Basic innovations of the software can be briefly characterized in the following way. Above all, the most general procedures of operation with *Maple* are intended for receiving information such as: number of the current *Maple* release, full paths to its basic subdirectories, current version of an underlying system, information about installed releases of *Maple*, etc. Similar tools are absent among the standard *Maple* software, however they appear for us useful enough at more advanced operation with the package *Maple*. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**CDM** - *basic directories of the package Maple*

**CCM**

**KCM**

Call format of the procedures:

**CDM()**

**CCM()**

**KCM()**

Formal arguments of the procedures: *no*

Description of the procedures:

The call **CDM()** returns the full path to the main directory of the package as a string.

The call **CCM()** returns the full path to the directory containing the package kernel as a string.

The call **KCM()** returns the name of the directory containing the package kernel as a string.

Along with the information of the built-in function **currentdir()** which returns the current *Maple* subdirectory of an underlying OS, in a whole series of cases the operating information about the root directory of the package and/or about directory with the package kernel can be needed. The above simple procedures solve these problems.

The procedure **CCM** has essential enough meaning, ensuring the correct information on a full path to important BIN-subdirectory of *Maple*. Really, at installation of *Maple 8* on platform *Windows 98SE* this subdirectory will have the name "BIN.W9X", whereas on platform *Windows XP Professional* -

"BIN.WIN". The given information allows to use more effectively a file structure of *Maple* at programming various appendices.

```
CDM:= proc ()
local cd, k, h, L;
  assign(L = [libname]);
  for k to nops(L) do
    if Search1(Path(L[k]), "\\lib", 'h') and h = ['right'] then return L[k][1..-5] else next end if
  end do
end proc
KCM:= proc () local a; assign(a = CCM()); a[Search2(a, {"\\", "/"})[-1]+1 .. -1] end proc
CCM:= proc()
local a, b, h, f;
  assign(f = "$Art_Kr$", b = "bin.w"), system(cat("DIR /A:D ", blank(CDM()), " > ", f));
  do h := readline(f);
    if h = 0 then remove(f); error "subdirectory BIN of Maple cannot be identified"
    else a := ewsc(Case(h), [b], [1]);
      if a <> [] then return remove(f, cat(CDM(), "\\ ", a[1]) end if
    end if
  end do
end proc
```

Typical examples of the procedures use:

```
> CDM(); ⇒ "C:\Program Files\Maple 8"
> CCM(); ⇒ "C:\Program Files\Maple 9\bin.win"
> KCM(); ⇒ "bin.win"
```

**Imaple** - *identification of the installed releases of the package Maple*

Call format of the procedure:

**Imaple()**

Formal arguments of the procedure: **no**

Description of the procedure:

In a series of cases for more effective operation with the package, the user can has on a PC four last releases 6, 7, 8 and 9 simultaneously. Such need arises, for example, at programming of software compatible relatively to different releases. In our books [29-33, 43, 44] the principal questions of actual incompatibility of release 6. on the one hand, and releases 7, 8, on the other hand, of the package *Maple* are discussed enough minutely.

The procedure call **Imaple()** returns the table whose indices are *Maple* release identifiers *Maple6*, *Maple7*, *Maple8*, *Maple9* whereas its entries define sequences of full paths to main *Maple* subdirectory "Bin.wnt", "Bin.W9X" or "Bin.win" for the corresponding release. Naturally, the returned table cannot be empty, whereas its empty entries defines absence of installed corresponding *Maple* releases. In addition, it is necessary to note, if in the same *Windows* environment several versions of 6th release or 6th and {7|8|9}th release can be installed, then the 7th and 8th releases do not admit multiple versions.

```
Imaple:= proc()
local a, b, d, k, L, j, h, f, t;
  assign(a = [Adrive()], f = "$ArtKr.157");
```

```
L := table(['Maple6' = NULL, 'Maple7' = NULL, 'Maple8' = NULL, 'Maple9' = NULL]);
for j in a do system(cat("Dir /S ", j, ":\ \ > ", f));
  do h := readline(f);
    if h = 0 then break else h := Case(h) end if;
    if search(h, "directory of ", 't') then d := h[t .. -1]
    elif search(h, "wmaple.exe") then L['Maple6'] := L['Maple6'], d[14 .. -1]
    elif search(h, "maplew.exe") then L['Maple7'] := L['Maple7'], d[14 .. -1]
    elif search(h, "maplew8.exe") then L['Maple8'] := L['Maple8'], d[14 .. -1]
    elif search(h, "cwmapple9.exe") then L['Maple9'] := L['Maple9'], d[14 .. -1]
    end if
  end do;
close(f)
end do;
for k in map(op, [indices(L)]) do b := [L[k]];
  seq('if (search(b[j], "\ \ prefetch"), assign('b' = delel(b, j)), NULL), j = 1 .. nops(b)); L[k]:=op(b)
end do;
fremove(f), eval(L)
end proc
```

Typical example of the procedure use:

```
> Imaple();
table(['Maple7' = "c:\program files\maple 7\bin.wnt",
      Maple8 = "c:\program files\maple 8\bin.win",
      Maple9 = "c:\program files\maple 9\bin.win",
      Maple6 = "c:\program files\maple 6\bin.wnt"])
```

It is necessary to note, the 8th release recognizes itself as the 7th release, whereas the 9th release recognizes itself as the 8th release.

**Lprot** - *protect names of the user and Maple libraries from modification*

Call format of the procedure:

**Lprot**{args}

Formal arguments of the procedure:

*args* - (optional) sequence of symbols

Description of the procedure:

The procedure call **Lprot**{args} returns the *NULL* value, i.e. nothing, assigning the attribute "protected" to all necessary names (symbols) of the user library and Maple. The procedure call **Lprot**() protects names such as:

*all, APPEND, Arguments, arity, assignable, assignable1, binary, BINARY, Break, boolproc, builtin, byte, bytes, chkpnt, coincidence, complex, complex1, false, gamma, infinity, true, Catalan, FAIL, Pi, correct, create, Cycle, Decimal, decimal, default, del, delel, delel1, delete, digit, dir, dirax, direct, display, element, Empty, End, Exit, exit\_type, exprseq, extract, file, file1, Fin, fpath, frame, globals, heap, hold, insensitive, insert, inside, invalid, left, letter, lex\_tab, libobj, list, list1, listlist, locals, lower, Lower, lowercase, LRM, mlib, Mod, mod1, Next, nestlist, nonsingular, NRM, only, package, path, prn, procname, pvl, RAW, READ, realnum, rem\_tab, Repeat, restore, right, rlb, Roman, Russian, sensitive, seqn, sequent, set, set1, setset, shape, Special, ssign, ssl, statement, sublist, Table, terminal, TEXT, toc2005, upper, Upper, uppercase, Variable, VGS, WRITE, variable*

and names of all procedures and modules from the user library containing the procedure *MkDir*.

While the procedure call *Lprot(args)* with optional actual *args* arguments additionally provides protection of names defined by the sequence of symbols and/or names of means of user libraries given by full path that are contained in predetermined variable *libname*. As a rule, the call of the given procedure is located in the initialization datafile "Maple.ini" after definition of the predetermined variable *libname*, for example:

"Maple.ini": libname:= "C:/Program Files/Maple 9/LIB/UserLib", libname: **Lprot()**:

Such approach allows to organize conveniently enough protection of necessary names and symbols in *Maple*. Procedure can be used and locally for operative protection against updating of the *Maple* objects. Another approach of labels protection is very simple and consists in the following. If a procedure uses, for example, labels **L1**, **L2**, ..., **Ln**, at the beginning of the procedure code, the procedure call **unassign('L1', 'L2', ..., 'Ln')** is coded. It allows to provide the reliable protection of all labels of procedure against their possible definition outside of a body of procedure since labels of procedures are global symbols concerning procedures.

```
Lprot := proc()
local a, b, c, d, k;
  assign(c = {}, d = []); if nargs <> 0 then for k in [args] do
    if member(CF(k), {map(CF, {libname})}) then
      d := [op(d), k]
    elif type(k, 'symbol') then c := {k, op(c)}
    end if
  end do
end if;
if type(MkDir, 'libobj') then a := [op(march('list', _libobj)), seq(op(march('list', k)), k = d)]
elif d <> [] then a := [seq(op(march('list', k)), k = d)]
else error "user libraries are absent in the predefined variable <libname>"
end if;
  protect('if' (c = {}, NULL, op(c)), `all`, `APPEND`, `Arguments`, `arity`, `assignable`, `assignable1`,
`binary`, `BINARY`, `Break`, `boolproc`, `builtin`, `byte`, `bytes`, `chkpnt`, `coincidence`, `complex`,
`complex1`, `false`, `gamma`, `infinity`, `true`, `Catalan`, `FAIL`, `Pi`, `correct`, `create`, `Cycle`,
`Decimal`, `decimal`, `default`, `del`, `delel`, `delel1`, `delete`, `digit`, `dir`, `dirax`, `direct`,
`display`, `element`, `Empty`, `End`, `Exit`, `exit_type`, `exprseq`, `extract`, `file`, `file1`, `Fin`,
`fpath`, `frame`, `globals`, `heap`, `hold`, `insensitive`, `insert`, `inside`, `invalid`, `left`, `letter`,
`lex_tab`, `libobj`, `list`, `list1`, `listlist`, `locals`, `lower`, `Lower`, `lowercase`, `LRM`, `mlib`, `Mod`,
`mod1`, `Next`, `nestlist`, `nonsingular`, `NRM`, `only`, `package`, `path`, `prn`, `procname`, `pvl`,
`RAW`, `READ`, `realnum`, `rem_tab`, `Repeat`, `restore`, `right`, `rlb`, `Roman`, `Russian`,
`sensitive`, `seqn`, `sequent`, `set`, `set1`, `setset`, `shape`, `Special`, `ssign`, `ssll`, `statement`,
`sublist`, `Table`, `terminal`, `TEXT`, `toc2005`, `upper`, `Upper`, `uppercase`, `Variable`, `VGS`,
`WRITE`, `variable`, seq('if' (b[1][1 .. 2] = ":-", NULL, cat(`, b[1][1 .. -3])), b = a))
end proc
```

Typical example of the procedure use:

```
> Lprot(AVZ, AGN, VSV, "C:/Program Files\\Maple 9/LIB\\UserLib");
> READ:= 2005;
```

Error, attempting to assign to **READ** which is protected

**prestart - clearing variables, outer relative to procedure**

Call format of the procedure:

```
prestart({, 'var' {, 1}})
```

Formal arguments of the procedure:

- var** - (optional) sequence of unevaluated variables
- 1** - (optional) defines mode of variables cleaning

Description of the procedure:

The **restart** command will cause the *Maple* kernel to clear its internal memory so that it acts as if you had just started *Maple*. However, the **restart** command will only work at the top level. It cannot be executed from within a procedure, or from inside a file being read by the **read** statement. This is because forgetting everything while something is executing would be very dangerous. Whereas in some cases it is necessary to clear the variables determined outside of procedure, from within the given procedure. Procedure **prestart** allows to solve the given problem.

The procedure call **prestart()** clears all variables, determined in the current session, excepting the basic variables of the package. The procedure call **prestart('x1', 'x2', 'x3', ..., 'xk')** clears all variables, determined in the current session, excepting the variables 'xj' (j=1..k) and the basic variables of the package. At last, the procedure call **prestart('x1', 'x2', 'x3', ..., 'xk', 1)** clears the variables 'xj' (j=1..k), determined in the current session, excepting the basic variables of the package.

In addition, it is necessary to have in mind, the procedure call always returns the set of variables that have been cleaned. Arguments (if they exist) are coded in unevaluated form, for example, 'V'. The procedure have a lot of useful appendices in programming.

```
prestart := proc()
local _avz_agn_asv_, f, h, k;
  assign(f = cat(currentdir(), "////Art_Kr$$$"),
    _avz_agn_asv_ = (((({anames()} minus {anames(`procedure`)})) minus
      {anames(`module`)})) minus {anames('environment')})) minus {anames('builtin')})) minus
      {anames('package')})) minus {'Context/InitProcs', 'ContextKey', 'type/interfaceargs',
      'ContextData', 'stack', 'lasterror', 'lastexception', 'context/InitProcs'});
  if 2 <= nargs and args[-1] = 1 then
    assign67(_avz_agn_asv_ = [args[1 .. -2]]), writeline(f, cat(cat(cat("unprotect(",
      seq(cat("", _avz_agn_asv_[k], "", k = 1 .. nops(_avz_agn_asv_))[1 .. -2], "):"),
      cat(cat("unassign(", seq(cat("", _avz_agn_asv_[k], "", k = 1 .. nops(_avz_agn_asv_))[1 .. -2], "):"),
      k = 1 .. nops(_avz_agn_asv_))[1 .. -2], "):"), close(f);
    read f;
    remove(f, {args[1 .. -2]})
  elif 0 <= nargs then _avz_agn_asv_ := _avz_agn_asv_ minus {args};
    if {_avz_agn_asv_} <> {} then _avz_agn_asv_ := op(_avz_agn_asv_);
      save _avz_agn_asv_, f; assign(h = SLD(readline(f)[18 .. -2], " , ")), close(f)
    end if;
    writeline(f, cat(cat(cat("unprotect(", seq(cat("", h[k], "", k = 1 .. nops(h))[1 .. -2], "):"),
      cat(cat("unassign(", seq(cat("", h[k], "", k = 1 .. nops(h))[1 .. -2], "):"),
      k = 1 .. nops(h))[1 .. -2], "):"), close(f);
    read f;
    remove(f, {_avz_agn_asv_})
  end if
end proc
```

Typical example of the procedure use:

```
> a, b, c, d, e, f, h, g, x, y, z, t:= 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12;
> protect('x', 'z', 'g'); prestart('a', 'b', 'x', 'g', 1);
      {g, a, x, b}
```

> a, b, c, d, e, f, h, g, x, y, z, t, prestart('c', 'h', 't');  
a, b, 3, 4, 5, 6, 7, g, x, 10, 11, 12, {f, y, z, e, d}

**Release** - check of the current release of the Maple package

**Release1**

Call format of the procedures:

**Release**({, 'h'})

**Release1**({, 'h'})

Formal arguments of the procedures:

**h** - (optional) an unevaluated name

Description of the procedures:

The procedure call **Release()** returns the integer 6, 7, 8 or 9 for four last releases of *Maple*; otherwise the value *Other release* is returned. If optional unevaluated actual **h**-argument (*name*) is used, via this name the full path to the main directory of the *Maple* package is returned as a *string*.

In contrast to the first procedure, the procedure call **Release1()** returns the integer 4, 5, 6, 7, 8 or 9 for six last releases of the package; otherwise the value *Other release* is returned. If optional unevaluated actual **h**-argument (*name*) is used, via this name the full path to the main directory of the package is returned as a *string*.

First of all, the given procedures represent the doubtless interest in connection with absence of full compatibility (both "from above - down" and "from below - upwards") of release 6, on the one hand, and releases 7, 8 and 9, on the other hand, of the package *Maple*; in a lot of cases the identification of an used current release of *Maple* is required that the above procedures provide.

**Release := proc()**

**local** k, cd, L, R;

assign(cd = currentdir(), L = [libname]);

**for** k **to** nops(L) **do**

**try** currentdir(cat(L[k][1 .. searchtext("/lib", L[k])], "license"))

**catch** "file or directory does not exist": NULL

**end try**

**end do;**

assign(R = readbytes("license.dat", "TEXT", `infinity`), close("license.dat"), `if` (nargs = 1, assign([args][1] = currentdir()[1 .. -9]), NULL), currentdir(cd);

`if` (search(R, "Maple7With"), 7, `if` (search(R, "MapleWith"), 6, `if` (search(R, "Maple8With"), 8,

`if` (search(R, "Maple9With"), 9, `if` (search(R, "Maple9.5With"), 9.5,

`if` (search(R, "Maple10With"), 10, `Other release`))))))

**end proc**

**Release1 := proc()**

**local** v, f, k;

assign(f = cat(currentdir(), "/\$Kr\_Art\$.m"), (**proc**(x) **local** v; v := 63; **save** v, x **end proc**)(f),

  `if` (nargs = 0, 9, assign(args[1] = [type(mkdir, 'libobj'), \_libobj][2])),

  assign(v = readbytes(f, "TEXT", `infinity`), remove(f);

  `if` (v[4] = "5", 5, `if` (v[4] = "4", 4, [search(v, "R0", 'f'),

  `if` (member(parse(v[2 .. f - 1]), {k \$ (k = 6 .. 10)}), parse(v[2 .. f - 1]), Release())[2]))

**end proc**

Typical example of the procedure use:

```

> Release();                                     # release 6
                                                6
> Release('h'), h;                             6, "C:\MAPLE 5 6 7\MAPLE_EDU\MAPLE 6 EDU"
> Release1();                                    6
> Release()                                     # release 7
                                                7
> Release('h'), h;                             7, "C:\MAPLE 5 6 7\MAPLE_EDU\MAPLE 7 EDU"
> Release1();                                    7
> Release();                                     # release 8
                                                8
> Release('h'), h;                             8, "C:\PROGRAM FILES\MAPLE 8"
> Release1();                                    7
> Release();                                     # release 9
                                                9
> Release('h'), h;                             9, "C:\Program Files\Maple 9"

```

Of the examples, represented above, follows, the results of the calls of procedures **Release()** and **Release1()** for releases 6 and 7 of *Maple* relatively to the returned release number are identical. Whereas for a case of the 8th release the calls of procedures **Release()** and **Release1()** return the value 8 and 7 accordingly. For the last 9th release both calls **Release()** and **Release1()** return the same value 9. The reason of the given fact is the following important circumstance. As it was already noted above, and it has been considered in detail in our books [29-33, 43, 44], releases 6 and 7 of the package are incompatible "from above - down" and "from below - upwards" relative to a lot of the important program parameters. In particular, releases 6 and 7 are different relative to internal format. Appreciably the problem of compatibility of these releases is solved by the software represented in this book and other our books [29-33, 43, 44].

Meanwhile, the last 9th release of the package relative to the program environment inherits all basic characteristics of the previous 8th release, having other numbering (though and in this case the question of full compatibility of releases 7, 8 and 9 should be the subject of separate consideration). Therefore, if the **Release** procedure returns the official number of the current release, then the **Release1** procedure returns the release number responding the current program environment of the package *Maple*.

Already from the previously mentioned, quite pertinently to make an essential conclusion, the developers of the package first should pay attention to elimination of many essential mistakes and defects of basic means of the previous releases, instead of increase of the numbering of its releases in commercial interests. The part of these defects the *above user library* (**AUL**) eliminates; the **AUL** contains means represented in the present book, and is logically linked to the main *Maple* library with the higher priority of the **AUL** at search of software.

**varsort** - *variables exchanging by their values*

Call format of the procedure:

**varsort**({args} [, sf])

Formal arguments of the procedure:

*args* - (optional) a sequence of unevaluated variables, for example, 'x', 'y', 'z', ...

*sf* - (optional) an ordering function

Description of the procedure:

In a series of cases, it is necessary to execute the variables exchanging by their values. The simplest example - exchange of values for variables in order their values have been at ascending order. For instance, variables *x*, *y* and *z* having values 62, 57 and 37 should receive values 37, 57 and 62 accordingly. The following procedure **varsort** solves higher problem even.

The procedure call **varsort('x', 'y', 'z', ...)** returns the *NULL* value, i.e. nothing, providing the exchange of variables *x*, *y*, *z*, ... by their values at ascending order. In addition, the exchange can hold only for groups of adjacent variables (*in a sequence of actual arguments*) with values of types {*realnum*, *symbol*} accordingly; undefined variables remain without change and ordering of values of *symbol*-type holds at lexicographically increasing order.

If optional *sf*-argument is given, it is used to define the ordering for a sorting of *realnum*-values; *sf*-argument must define a Boolean-valued function of two arguments. Specifically, **sf(x, y)** should return the *true* value if *x* precedes *y* in the desired ordering. Otherwise, it should return the *false* value. By varying order of actual arguments at the procedure call, one can receive higher features of the above-described variables exchanging by their values. The procedure handles basic especial and erroneous situations.

```
varsort:= proc()  
local a, b, c, d, k;  
  if nargs = 0 then return else assign(c = []) end if;  
  d:= proc(x, y)  
    try  
      if evalf(x) < evalf(y) then true else false end if  
    catch: `if (map(type, {x, y}, 'symbol') = {true}, lexorder(x, y), true)  
    end try  
  end proc;  
  a:= [seq(`if (type(args[k], 'boolproc'), assign('b' = args[k]), `if (type(args[k], 'symbol'),  
    op(args[k], assign('c' = [op(c), k])), NULL)), k = 1 .. nargs)];  
  a:= sort(a, `if (type(b, 'boolproc'), b, d));  
  for k to nops(a) do assign(args[c[k]] = a[k]) end do  
end proc
```

Typical examples of the procedure use:

```
> varsort(57, 8, 15, 37, 41, 62), varsort();  
> m:=vsv: n:=avz: p:=agn: varsort('m', 'n', 'p'), m, n, p; => agn, avz, vsv  
> x:=57: y:=37: varsort('x', 'y'), x, y; => 37, 57  
> a:=8: b:=15: c:=57: varsort('c', 'a', 'b'), a, b, c; => 15, 57, 8  
> x:=62: y:=57: z:=37: m:=vsv: n:=avz: p:=agn: varsort('x','y','z','m','n','p'), x,y,z,m,n,p;  
37, 57, 62, agn, avz, vsv  
> x:=62: y:=57: z:=37: m:=vsv: n:=avz: p:=agn: varsort('x','z','y','m','p','n'), x,y,z,m,n,p;  
37, 62, 57, agn, vsv, avz  
> x:= 2005: z:= 65: h:= 350: y:= 1995: g:= 58: s:= 38: varsort('x','z','h','y'), x, z, h, y, g, s;  
65, 350, 1995, 2005, 58, 38
```

The **varsort** procedure is rather broadly used in numerous problems dealing with various variables processing.

**parvar** - generating of undefined number of assignments

Call format of the procedure:

**parvah(P::symbol)**

Formal arguments of the procedure:

**P** - symbol or name

Description of the procedure:

In a whole series of cases it is necessary to execute assignments of *Maple*-expressions to variables whose number is beforehand not known and which is determined as a result of some calculations, for example, cyclic character. The given problem is solved by simple enough *parvar* procedure. Generally, the procedure call has at least one actual argument **P** that is expression of *symbol*-type. As other optional arguments any *Maple*-expressions are permitted. Moreover, *NULL* value can act as actual value for **P**-argument, i.e. nothing.

The procedure call *parvar()* returns the empty list, i.e. []; whereas the procedure call *parvar(P)* returns the 2-element list whose first element defines number of the calculated variables (**1**) and the second defines a variable of the form **cat('\_, P)** to which the *NULL* value is assigned, i.e. nothing. At last, the procedure call *parvar(P, x1, ..., xk)* returns (**k+1**)-element list whose first element defines number of the calculated variables (**k**) and the others define names of variables of the form **cat('\_, P, h)** to which expressions **xj (h=1.. k)** are assigned accordingly. Examples of the fragment presented below illustrate application of the procedure and use of the calculated variables.

```
parvar := (X::symbol) -> `if`(nargs = 0, [], `if`(nargs = 1, op([[1, cat('_, X)], assign67(cat('_, X) =
NULL)]), op([[nargs - 1, seq(cat('_, X, k), k = 1 .. nargs - 1)], seq(assign(cat('_, X, k) =
args[k + 1]), k = 1 .. nargs - 1)]))
```

Typical examples of the procedure use:

```
> k:=58: `__`:=63: parvar(), parvar(H); => [], [1, __H]
> [__H]; => []
> parvar(H, [a,b], sin(x), sqrt(c+d), 63); => [4, __H1, __H2, __H3, __H4]
> %[2 .. 5]; => [[a, b], sin(x), (c + d)^(1/2), 63]
> G:=[]: for k to 15 do G:=op(G), k end do: parvar(S, op(G));
[15, __S1, __S2, __S3, __S4, __S5, __S6, __S7, __S8, __S9, __S10, __S11, __S12, __S13, __S14, __S15]
> %[2 .. 16], __S9, __S15; => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], 9, 15
```

**Users** - Maple serial number and full path to the **USERS** directory

Call format of the procedure:

**Users(h {, t})**

Formal arguments of the procedure:

**h** - an arbitrary name

**t** - (optional) an arbitrary valid *Maple* expression

Description of the procedure:

The **USERS** directory of *Maple* contains at least the *Maple* initialization files of the different level. Information of these files is useful for many appendices. The procedure call **Users(h)** returns full path to the **USERS** directory in a format allowable for correct use by the standard *Maple* means of access to datafiles, and through actual **h** argument additionally returns the serial number of the current *Maple* installation. At absence of serial number, the **h** argument is returned unevaluated. If the optional second argument **t** has been encoded, then full path to the **USERS** directory is returned in a format allowable for correct use by the functions *system* and *ssystem*. The procedure is a rather useful at programming of access to datafiles of different types and in a lot of other appendices..

```

Users:= proc(h::evaln)
local a, b, c;
  a := fopen(cat(CCM(), "\\MapleSys.ini"), 'READ', 'TEXT');
  while not Fend(a) do b := readline(a);
    if b[1 .. 7] = "UserDir" then c := `if`(1 < nargs, Path(b[15 .. -1]), b[15 .. -1])
    elif b[1 .. 6] = "Serial" then unprotect('h'), assign('h' = SD(b[14 .. -1]))
    end if
  end do;
  close(a), c
end proc

```

Typical examples of the procedure use:

> **Users(h), h, Users(t, 7), t;**

"C:\Program Files\Maple 9\Users", 917864204, "c:\program files\maple 9\users", 917864204

> **DirF(Users(h));**

["c:\program files\maple 9\users\maple9.ini", "c:\program files\maple 9\users\links.mws",  
"c:\program files\maple 9\users\maple.ini"]

### **mapleacs – testing of Maple releases loaded in the current Windows session**

Call format of the procedure:

**mapleacs()**

Formal arguments of the procedure: *no*

Description of the procedure:

It is well-known, that various *Maple* releases are incompatible relative to a lot of important parameters, including basic means. It requires the simultaneous presence of several *Maple* releases as various *Windows* applications in a series of cases (*programming and debugging of compatible procedures and modules, etc.*). *Maple 9* represents a typical example of incompatibility even within of the same release.

*Maple 9* provides users with two worksheet interfaces. Both have access to the full mathematical engine of *Maple 9* and take advantage of the new functionality in *Maple 9*. By default, worksheets open in the enhanced and more modern Standard Worksheet. The Classic Worksheet has the traditional *Maple* worksheet look and uses less memory. User can change the *Maple* worksheet file association using the Worksheet File Association Selector application from the Tools folder of the *Maple 9* Start menu. The classical mode provides kernel '*cwmaple9.exe*', whereas the standard mode - kernel '*maplew9.exe*'. Meanwhile, both modes *Maple 9* are not fully compatible already relative to the built-in functions.

Simple check reveals, call of the built-in function '*system*' in a case (A) is correct with creation of a required file "C:\\temp\\tlistA.txt" whereas in a case (B) the call of the same function also gives a return code 0 (zero), however, the required file "C:\\temp\\tlistB.txt" does not create. Hence, a procedure, programmed and debugged in the first mode of *Maple 9* with use of the given function, will not be correct in the second mode of *Maple 9*, what contradicts the elementary requirements claimed to qualitative software. See fragment below. In this connection, the following question arises: If in the current *Windows* session the few *Maple* releases are simultaneously loaded, how a procedure can define the loaded releases and the active *mws*-files accompanying them?

The following procedure *mapleacs* solves this problem. Successful procedure call *mapleacs()* returns the nested list of sublists with {2|3} elements depending on *Maple* release. The first element of the sublist defines the package release and its clone. So *Maple9s* defines *Maple 9* for kernel

'*maplew9.exe*' (standard mode) and *Maple9* for kernel '*cwmaple9.exe*' (classical mode). The second element defines a name or full path to an active document, whereas the third element (*if is*) defines the number of the server. In addition, the order of sublists in the list corresponds to the order of loading of *Maple* releases in *Windows*.

From the above-said follows, the procedure *mapleasc* correctly operates with releases 6 - 9; in addition, in *Maple 9* the classical mode is allowable only. The procedure implementation essentially uses the approach supported by two our procedures *com\_exe1* and *com\_exe2*, and built-in function *system*. The procedure *mapleasc* is rather useful at program processing of current *Maple* documents on the assumption of the few loaded *Maple* releases in the current *Windows* session.

```

mapleasc := proc()
local a, b, c, d, q, h, k, p, t, r, v;
  assign(a = interface(warnlevel), b = cat([libname][1], "\$$$Avz_Agn$$$"), h = [], p = []);
  assign67(interface(warnlevel = 0)), com_exe2({'tlist.exe'}, system(cat("tlist > ", Path(b)));
  do d := readline(b);
    if d = 0 then break
    elif Search2(Case(d), map(cat, {'maplew8', 'java', 'maplew', 'wmaple', 'cwmaple9'}, `exe`))
      <> [] then h := [op(h), d]
    end if
  end do;
  for k in h do
    if search(k, "java.exe", 't') then
      if not search(k, ["Server ", 'q']) then p := [op(p), ['Maple9s', [Suffix(k[t+8..-1], " ", v, 'del'),v][2]]]
      else p := [op(p), ['Maple9s', cat(``,[Suffix(k[t + 8 .. q - 4], " ", v, 'del'), v][2]), cat(``, k[q+1..-2])]]
      end if
    else
      search(k, "-", 't'), assign('d' = SLD(Red_n(k[1 .. t - 3], " ", 2), " ")), assign('d' = cat(d[3], d[4]));
      assign('r' = k[t + 2 .. -1]), `if` (d[-1] = "6", assign67('r' = r[2 .. -2], 'q' = NULL),
        [search(r, ["Server ", 't'), assign('r' = r[2 .. t - 4], 'q' = r[t + 1 .. -3])]);
      p := [op(p), map(convert, [d, r, q], 'symbol')]
    end if
  end do;
  interface(warnlevel = a), fremove(b), p
end proc

```

Typical examples of the procedure use:

```

> system("tlist > c:\\temp\\tlistA.txt"); => 0 # Classic Worksheet mode - Maple 9 - (A)
> type("c:\\temp\\tlistA.txt", 'file'); => true
> system("tlist > c:\\temp\\tlistB.txt"); => 0 # Standard Worksheet mode - Maple 9 - (B)
> type("c:\\temp\\tlistB.txt", 'file'); mapleasc(); => false
[[Maple9, rans_ian.mws, Server 1], [Maple6, vgtu47.mws], [Maple8, mapleasc.mws, Server 1]

```

**nvalue** – name determination on the basis of value allocated to it

Call format of the procedure:

**nvalue**({A})

Formal arguments of the procedure:

A – (optional) an arbitrary valid *Maple* expression

Description of the procedure:

In a series of cases on the basis of certain value it is necessary to determine a name (or names) to which in the current *Maple* session the given value has been allocated. This problem is solved by the *nvalue* procedure, basing on standard procedure *anames* and use of some peculiarities of the package. The procedure *anames* returns an expression sequence of names that are currently assigned values other than their own name. Thus, procedure reflects only the values of global variables received by them in the current session. The clause **restart** brings value *anames()* into initial status, namely: **restart; {anames()} -> {}** (for *Maple 6 and 7*) and **restart; {anames()} -> {'type/interfaceargs', interface}** (for *Maple 8 and 9*).

The procedure call *nvalue(A)* returns the list of global names to which in the current *Maple* session the value *A* has been allocated, whereas the call *nvalue()* returns the list of global names to which in the current session the *NULL* value has been allocated. The procedure call returns the empty set, i.e. {}, at absence of such names. Examples below well illustrate the said and do not demand any special elucidations.

```

nvalue := proc(A::anything)
local a, b, c, d, f, k, t, h;
  assign(a = (b -> `if` (search(cat("", b), "RTABLE_SAVE/"), NULL, b)),
    d = {}, f = cat(currentdir(), "/$Art_Kr$"));
  if nargs = 0 then {seq(`if` (eval(cat(`, k)) = NULL, cat(`, k), NULL),
    k = map(convert, {anames()}, 'string'))}
  elif whattype(args) = 'exprseq' then a := map(convert, {anames()}, 'string');
    {seq(`if` ([eval(cat(`, k))] = [args], cat(`, k), NULL), k = a)}
  elif type(A, `module`) then
    assign('a' =sstr(["\n" = NULL], convert(A, 'string')), 'b' = {anames(`module`)});
    for k to nops(b) do
      (proc(x) save args, f end proc)(b[k]);
      assign('c' =sstr(["\n" = NULL], readbytes(f, `infinity`, 'TEXT')[1 .. -3])),
        search(c, ":", 't'), remove(f);
      if c[t + 3 .. -1] = a then d := {op(d), cat(`, c[1 .. t - 2])}
      end if
    end do;
    d
  elif type(eval(A), {`module`, range, table, Matrix, Vector, matrix, vector, Array, array, function})
    then map(a, {seq(`if` (convert(eval(k), 'list') = convert(eval(A), 'list'), k, NULL),
      k = {anames(whattype(eval(A))))})}
  else {seq(`if` (evalb(eval(k) = A), k, NULL), k = {anames(whattype(eval(A))))}
  end if
end proc

```

Typical examples of the procedure use:

```

> a:=63: b:=63: c:="ransian63": c9:="ransian63": d:=table([1=63,2=58,3=38]): assign('h'='svegal'):
  t47:=19.42: t42:=19.42: L:=[x,y,z]: R:=a+b*I: B:=a+b*I: Z:=(a1+b1)/(c1+d1): f:=cos(x): g:=proc()
  `+(args)/nargs end proc: r:=x..y: m:=x..y: n:=x..y: Lasnamae:= NULL:
> Tallinn:=NULL: Grodno:=NULL: Vilnius:=NULL: Moscow:= NULL:
> nvalue(63), nvalue("ransian63"), nvalue(table([1=63,2=58,3=38])), nvalue(svegal), nvalue(19.42),
  nvalue([x,y,z]), nvalue(a+b*I), nvalue((a1+b1)/(c1+d1)), nvalue(cos(x)),
  nvalue(proc () `+(args)/nargs end proc), nvalue(x..y), nvalue();

```

{a, b}, {c, c9}, {d}, {h}, {t47, t42}, {L}, {R, B}, {Z}, {f}, {g}, {r, n, m}, {Tallinn, Grodno, Vilnius, Moscow, Lasnamae}

### System - expansion of the standard built-in system-function

Call format of the procedure:

**System(com)**

Formal arguments of the procedure:

*com* - a command that must be passed the host operating system

Description of the procedure:

The built-in function *system* is incorrectly being executed in the standard mode of *Maple 9* and above. The following procedure *System*, being analogue of the above *system*-function, eliminates a number of shortcomings inherent in it and is correctly being executed on the majority of DOS commands. Successful completion of *System*-procedure returns *zero* value, unsuccessful - **1**, whereas at impossibility to accept to execution a command, passed to it, the procedure initiates erroneous situation. The reason of the given situation can consist both in impossibility to execute DOS command, and in its absence in the operational environment of a computer.

Our library uses standard system-function with such DOS commands as *Attrib*, *Dir*, *Date (cdt)*, *Del*, *Deltree*, *Copy*, *Ren (Dir\_ren, F\_ren, RegMW, D\_ren)*, *Ver (winver, BootDrive, WD)*, *Vol*. The *System*-procedure successfully processes those DOS commands behind which not are in brackets of procedure of the library; these procedures at our approach are not being executed in standard mode of *Maple*. Whereas in other cases the replacement in procedures of *system*-function onto *System*-procedure does them available and in the standard mode.

```
System := proc(S::{string, symbol})
```

```
local a, b, c, d, t;
```

```
  assign(a = sstr(["/" = "\\\"], Case(cat("", S))), search(a, ">", b), assign(c = ssystem(a));
```

```
  if c = NULL then error "Maple can't execute command <%1>; perhaps, <%3> is absent in DOS",  
    a, search(a, " ", t), a[1 .. `if` (type(t, 'assignable1'), -1, t - 1)]
```

```
  elif c[1] = 0 and type(b, 'assignable1') then `if` (c[2] = "", 0, c[2])
```

```
  elif member(c[1], {0, 1}) and not type(b, 'assignable1')
```

```
  then assign('c' = ssystem(FNS(a[1 .. b - 1], " ", 2)), d = pathtf(FNS(a[b + 1 .. -1], " ", 1)));  
    writebytes(d, c[2]); close(d), 0
```

```
  else 1
```

```
  end if
```

```
end proc
```

Typical examples of the procedure use:

```
> CCM();
```

```
# Maple 9 - классический режим
```

```
"C:\Program Files\Maple 9\bin.win"
```

```
> CCM();
```

```
# Maple 9 - стандартный режим
```

```
Error, (in readline) file or directory does not exist
```

```
> CCM();
```

```
# Maple 9 - стандартный режим
```

```
"C:\Program Files\Maple 9\bin.win"
```

```
> System("DelTree D:/temp/aaa\\bbb\\ccc/ddd");
```

```
Error, (in System) Maple can't execute command <deltree d:\temp\aaa\bbb\ccc\ddd>;  
perhaps, <deltree> is absent in DOS
```

```
> System("attrib C:\\Temp\MANUR\\Tallinn\Grodno.asp");
```

```
"A SHR C:\temp\manur\tallinn\grodno.asp"
```

**occur** – check of occurrence of a subexpression into the given Maple-expression

Call format of the procedure:

**occur(X, Y, Z::evaln {, T})**

Formal arguments of the procedure:

**X, Y** – dismissible Maple-expressions

**Z** – assignable name

**T** – (optional) Maple-type (can be {function, procedure})

Description of the procedure:

Procedure **occur(X, Y, Z)** generalizes the standard means *has*, *hasfun*, *numboccur*, *depends*, allowing to do testing of the fact of occurrence of subexpression **X** into Maple-expression **Y** both relative to a type {function, procedure} of **X**-subexpression, and irrespectively to the type. The procedure call **occur(X, Y, Z)** returns *true*-value if **X** is subexpression of Maple-expressions **Y**, and *false*-value otherwise. In addition, through the third **Z**-argument the number of such occurrences is returned. Whereas the procedure call **occur(X, Y, Z, T)** with the fourth optional **T**-argument accepting value from set {function, procedure}, makes the above testing relative to the type {function, procedure} of **X**-subexpression.

```

occur := proc(X, Y, Z::evaln)
local a, b, c, d, t, h;
  if nargs = 3 or nargs = 4 and not member(args[4], {'procedure', 'function'}) then
    op(map(assign, [Z, a], numboccur(Y, X))), `if` (a = 0, false, true)
  else assign(a = cat([libname][1], "\_Art16Kr9\$_"), h = [],
    t = ((s) -> `if` (s[1] = "", cat(`, s[2 .. -2]), cat(`, s)))));
    writeto(a), dismantle(Y), writeto('terminal');
  do c := readline(a);
    if c = 0 then break elif not (FNS(c, " ", 1)[1 .. 9] = "FUNCTION(")
    then next
    else assign('c' = FNS(readline(a), " ", 1), search(c, ":", 'b');
      h := [op(h), c[b + 2 .. `if` (search(c, "#[protected", 'd'), d - 2, -1)]];
    end if
  end do;
  remove(a), op(map(assign, [Z, 'a'], numboccur(map(t, h), X))), `if` (a = 0, false, true)
end if
end proc

```

Typical examples of the procedure use:

> Res := (ln(x) + ln(y) - G(x, y, z) + G + diff/V)/(sqrt(G) - diff(G(x), x) + int(V(y), y) + ln(cos));

$$Res := \frac{\ln(x) + \ln(y) - G(x, y, z) + G + \frac{diff}{V}}{\sqrt{G} - \left(\frac{d}{dx} G(x)\right) + \int V(y) dy + \ln(\cos)}$$

> [occur(ln, Res, p, 'procedure'), p], [occur(diff, Res, p, 'function'), p]; ⇒ [true, 3], [true, 1]

> [occur(ln, Res, p), p], [occur(diff, Res, p), p]; ⇒ [true, 3], [true, 2]

> [occur(G, Res, p, 'procedure'), p], [occur(V, Res, p, 'function'), p]; ⇒ [true, 2], [true, 1]

> [occur(G, Res, p), p], [occur(V, Res, p), p]; ⇒ [true, 4], [true, 2]

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

## Chapter 2.

# Software for work with procedural and modular *Maple* objects

In the given chapter, a group of tools extending the possibilities of the package *Maple* of releases 6-10 at operation with procedures and program modules is represented. These tools support kinds of processing such as: converting of modules into procedures; testing of presence in files of incorrect modules; check of parameters of procedures and modules; check of activity (*availability for direct use*) of a procedure or a module; check of type of a modular table; converting of files of the input *Maple* format containing modules; converting of a module of the second type into the first; converting of a file of the input *Maple* format into a file of the internal *Maple* format, and vice versa, etc. The represented tools provide many of the manifold useful operations with procedural and modular objects of *Maple*. These tools are used enough widely at advanced programming of various problems in the *Maple*. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**dcemod** – a dynamic call of a module export

Call format of the procedure:

**dcemod**(*M*, *ex* {, *args*})

Formal arguments of the procedure:

**M** – symbol defining a module name

*ex* – symbol defining a name of a variable exported by the module **M**

*args* – (*optional*) an arguments sequence passed to the module export *ex*

Description of the procedure:

The *Maple* calls mechanism does not allow to use the dynamically generated names for a variable exported by a module. The following simple example well illustrates the told, namely:

```
> 3*SimpleStat:- SR([61, 56, 36, 40, 7, 14], 1, 10);
```

107

```
> M:=SimpleStat: ex:=SR: 3*M:- ex([61, 56, 36, 40, 7, 14], 1, 10);
```

```
Error, module does not export `ex`
```

However, in a lot cases this circumstance is a rather essential shortcoming leading to complication of programming of problems dealing with modules. The procedure **dcemod** lifts this restriction.

The procedure **dcemod**(*M*, *ex* {, *args*}) has at least two formal arguments whose purpose consists in the following: **M** – a module name, *ex* – name of a variable exported by the module **M**, and *args* – optional arguments passed to an export *ex*. If the procedure call contains only two actual arguments, then the call *ex*() is supposed. Successful procedure call **dcemod**(*M*, *ex* {, *args*}) returns a result of call **M:- ex**(*args*). The procedure handles all basic erratic and especial situations and in numerous appendices has shown itself as a very useful tool.

```

Dcemod:= proc(M::symbol, ex::symbol)
local a;
global _v61g56;
  `if`(nargs < 2, ERROR("quantity of arguments should be more than 1 but has been received
  <%1>", nargs), `if`(type(M, `module`), `if`(member(ex, {exports(M)}),
  assign67(a = "_$EE_", _v61g56 = NULL), ERROR("<%1> does not export <%2>", M, ex)),
  ERROR("<%1> is not a module", M)), null(writeline(a, cat("_v61g56`:=", M, "[", ex, "]("),
  `if`(2 < nargs, seqstr(args[3 .. nargs]), ``), "):"))),
  (proc() close(a); read a; remove(a) end proc)(), _v61g56, unassign('_v61g56')
end proc

```

Typical examples of the procedure use:

> **dcemod(ACC);**

Error, (in **dcemod**) quantity of arguments should be more than 1 but has been received <1>

> **dcemod(RANS, ian);**

Error, (in **dcemod**) <RANS> is not a module

> **dcemod(ACC, ian);**

Error, (in **dcemod**) <ACC> does not export <ian>

> **dcemod(ACC, Ds, [61,56,36,40,7,14], 1, 10);** ⇒ 360.400000

> **dcemod(ACC, Ds);**

Error, (in **ACC:-Ds**) **ACC:-Ds** uses a 3rd argument, t, which is missing

> **3\*dcemod(SimpleStat, SR, [61,56,36,40,7,14], 1, 10);** ⇒ 107

> **M:=SimpleStat: ex:=SR: M:- ex([61,56,36,40,7,14],1,10);**

> **M:- ex([61,56,36,40,7,14],1,10);**

Error, module does not export `ex`

**ismLib** - *check of availability of a procedure or program module*

Call format of the procedure:

**ismLib(P {, 'h'} {, '\$'})**

Formal arguments of the procedure:

**P** - symbol defining name of a procedure or program module

**h** - (*optional*) an unevaluated name

**`\$`** - (*optional*) key symbol

Description of the procedure:

The procedure **ismLib(P {, 'h'} {, '\$'})** returns the *true* value, if a procedure or program module whose name is specified by the first actual **P** argument is in one of libraries logically linked with the main package library, including main library also; otherwise the *false* value is returned. At coding of the second optional **h** argument through it the full path to the first library in a chain of libraries defined by the *libname* variable of the package, in which a program object, indicated by the **P** argument has been detected, is returned.

At last, at coding of the optional symbol **`\$`** as the last argument, the procedure **ismLib** supports a searching of the required procedure or program module in all libraries reflected in the predefined *Maple* variable *libname*, excluding the main package library.

```

ismLib:= proc(P::{name, symbol})
local sl, sm, k, j, p, h, LIB;
  assign(h = map(CF, [libname])), `if`(not member(`$`, {args})), assign(LIB = h),
  assign(LIB = [seq(`if` (h[p][-4 .. -1] = "\\lib", NULL, h[p]), p = 1 .. nops(h))]);

```

```

for k to nops(LIB) do
  assign('sl' = march(list, LIB[k])), assign('sm' = map2(op, 1, sl));
  if member(cat("", P), {seq(sm[j][1 .. -3], j = 1 .. nops(sm))}) then
    RETURN(true,
      `if` (2 <= nargs, assign(args[2] = LIB[k], NULL))
    else NULL
  end if
end do;
false
end proc

```

Typical examples of the procedure use:

> **ismLib(ParProc1, 'h'), h;**

*true, "c:\program files\maple 9\lib\userlib"*

> **ismLib(Read1, 'h'), h, ismLib(MkDir, 't'), t;**

*true, "c:\program files\maple 9\lib\userlib", true, "c:\program files\maple 9\lib\userlib"*

> **ismLib(assign, 'p', '\$'), p; ismLib(assign, 'p'), p;**

*false, p  
true, "c:\program files\maple 9\lib"*

**IO\_proc** - *check of a procedure to be an iolib function*

Call format of the procedure:

**IO\_proc(P)**

Formal arguments of the procedure:

**P** - symbol or positive integer defining name or number of an iolib function accordingly

Description of the procedure:

The procedure call **IO\_proc(P)** returns the *number* of an *iolib* function specified by its **P** name, or the *name* of an *iolib* function specified by its **P** number; otherwise, an appropriate error arises. The procedure does not provide check of *iolib* functions of the **process** package of Maple 6 on Windows platform, because the package is supported only under UNIX and UNIX-compatible systems.

**IO\_proc:= proc(P::{posint, symbol})**

**local** a, k, p, T\_iolib, omega, h;

```

  assign(T_iolib = table([currentdir = 27, fclose = 7, close = 7, feof = 21, fflush = 23, fprintf = 9,
    fremove = 8, fscanf = 11, getenv = 28, iostatus = 13, march = 31, mkdir = 29, open = 20,
    readbytes = 4, readline = 2, rmdir = 30, sprintf = 10, sscanf = 12, writebytes = 5, exec = 17,
    block = 19, fork = 16, kill = 22, launch = 36, pclose = 7, pipe = 15, popen = 14, wait = 18,
    "process:-exec" = 17, "process:-block" = 19, "process:-fork" = 16, "process:-kill" = 22,
    "process:-launch" = 36, "process:-pclose" = 7, "process:-pipe" = 15, "process:-popen" = 14,
    "process:-wait" = 18]), assign(omega = (() -> `if` (nargs = 1, args, [args]));
  `if` (type(P, posint), `if` (member(P, {2, 4, 5, 36, p $ (p = 27 .. 31), k $ (k = 7 .. 23)}),
    map(convert, omega(RTab(T_iolib, P)), symbol),
    ERROR("<%1> is invalid number for an iolib Maple function", P)), [ `if` (search(P, "-"),
    assign(h = convert(P, string)), assign(h = P)), assign(a = T_iolib[h]), `if` (type(a, posint),
    RETURN(a), ERROR("<%1> is not an iolib function", P))]

```

**end proc**

Typical examples of the procedure use:

```
> IO_proc(Mkdir); IO_proc(RANS); IO_proc(process:- exec);
Error, (in IO_proc) <Mkdir> is not an iolib function
Error, (in IO_proc) <RANS> is procedure of a package or program module
or is not a procedure
```

17

```
> map(IO_proc, [close, mkdir, iostatus, 8, 27, readbytes, 13, 21, 7, writebytes, 8]);
[7, 29, 13, remove, currentdir, 4, iostatus, feof, [fclose, process:-pclose, pclose, close], 5, remove]
> x:= 2005: z:= 65: h:= 350: y:= 1995: g:= 58: s:= 38: varsort('x', 'z', 'h', 'y'), x, z, h, y, g, s;
65, 350, 1995, 2005, 58, 38
```

**IOproc** – check of Maple procedures upon conformity to iolib-functions

Call format of the procedure:

**IOproc(P {, 'h'})**

Formal arguments of the procedure:

**P** – an arbitrary Maple procedure

**h** – (optional) an assignable name

Description of the procedure:

The procedure call **IOproc(P)** returns *true* value if Maple procedure determined by actual **P** argument, contains calls of *iolib*-functions, and *false* value otherwise. Whereas the procedure call **IOproc(P, 'h')** with two arguments through the second argument **h** in addition allows to receive the list of numbers of *iolib*-functions used by the procedure **P**.

```
IOproc := proc(P::procedure)
local a, b, c, k, t;
  assign(a = sub_1(["\n" = NULL, "\t" = NULL], convert(eval(P), 'string')), c = {}),
  assign(b = Search2(a, {" iolib("}));
if b = [] then false else seq(assign('c' = {op(c), parse(a[k + 7 .. k + [search(a[k + 7 .. -1], "", 't'), t][2]
+ 5)]}), k = b) end if;
  true, (proc() if nargs <> 1 then if type(args[2], 'assignable1') then
    assign(args[2] = `if` (nops(c) = 1, c[1], c))
    else error "2nd argument must be assignable name but had received <%1>", args[2]
    end if end if end proc)(args)
end proc
```

Typical examples of the procedure use:

```
> map(IOproc, [readbytes, readline, iostatus, filepos, feof, writebytes, fopen, open, close]);
[true, true, true, true, true, true, true, true, true]
> [IOproc(feof, 'h'), h], [IOproc(readline, 'h'), h], [IOproc(iostatus, 'h'), h], [IOproc(open, 'h'), h];
[true, 21], [true, 2], [true, 13], [true, 20]
```

**ParProc** – check of parameters of a procedure, program module or package

**ParProc1**

**Sproc**

Call format of the procedures:

**ParProc(P)**

**ParProc1(P {, 'h'})**

**Sproc(P {, 'h'})**

Formal arguments of the procedures:

**P** - symbol defining name of a procedure, program module or package

**h** - (*optional*) an unevaluated name

Description of the procedures:

The important element of work with procedures and program modules is check of their parameters (*arguments, local and global variables, options, the exported variables, etc.*). The procedures represented below solve this problem to a certain extent. The procedure call *ParProc(P)* returns the 1-dimensional *column*-array whose elements have the following view:

*<type of parameter> = (<a sequence of values>)*

for a procedure specified by its name **P** (*a library procedure or procedure activated in the current session*). For the built-in functions and functions of the I/O library, their numbers are returned. If the argument **P** specifies name of a program module (active or library module), the list of variables exported by the program module is returned.

The *ParProc1* procedure represents an useful extension of the previous procedure, providing the more detailed analysis of parameters of procedures and program modules. Similarly, to the previous, the procedure call *ParProc1(P)* returns a 1-dimensional *column*-array of values of parameters for a procedure or program module whose name is specified by the first actual **P** argument. If the second optional **h** argument has been coded, then through it the set of full paths to libraries containing the given program object is returned. Additionally, the appropriate warning about location of sought object is output.

At last, the procedure call *Sproc(P)* returns the *true* value if a procedure or program module specified by the actual **P** argument is located in a library logically linked with the main *Maple* library, including the main library also; otherwise the *false* value is returned. If the second optional **h** argument has been coded and the procedure call returns the *true* value, then through **h** the 3-elements list or 4-elements list of the following kinds are returned, namely:

[*Proc*, {*built-in* | *iolib*}, *<a function number>*, *<full path to the Maple library>*]

[*{Proc | Mod | package}*, {*Maple | User | Maple&User*}, *<set of full paths to libraries>*]

The first element of both lists defines the type of a program object {*Proc* - procedure | *Mod* - module | *package* - package}, whereas the others are defined by the implementation kind of a program object, namely: *built-in*, *iolib* or *library*. For the first two kinds the 4-elements list is returned whose the second element defines the implementation kind {*built-in* | *iolib*}, the third element - a function number, and the fourth - full path to the main *Maple* library. Whereas for all implementation kinds the procedure call returns the 3-elements list whose the second element defines the type of library containing a program object {*Maple* - the main *Maple* library | *User* - the user library | *Maple&User* - the main *Maple* and user libraries}, the third element - set of full paths to libraries with a specified program object.

**ParProc:= proc(P::symbol)**

**local** k, L, R, h;

**option** *system, remember*, `Copyright International Academy of Noosphere - Tallinn - 2000`;

L := [Arguments, locals, `options`, rem\_tab, `description`, globals, lex\_tab, exit\_type];

assign(R = Release1()), `if (P = randomize, goto(A), NULL), `if (type(P, `module`),

RETURN(cat(P, `is module with exports`, [exports(P)]), `if (type(P, procedure),

assign(h = Builtin(P)), ERROR("%1> is not a procedure and not a module", P));

**if** type(h, integer) **then** RETURN(`builtin function`, h)

**else try** h := IO\_proc(P); **if** type(h, integer) **then** RETURN(`iolib function`, h) **end if**

**catch** "": NULL **end try**

```

end if;
A;
array(1 .. add(`if` (op(k, eval(P)) = NULL, 0, 1), k = 1 .. R + 1), 1 .. 1,
  [seq(`if` (op(k, eval(P)) <> NULL, [L[k] = op(k, eval(P))], NULL), k = 1 .. R + 1)])
end proc
ParProc1:= proc(M::{procedure, `module`})
local a, b, c, d, p, h, t, z, cs, L, R, N, omega, nu;
global `_62`, ParProc, Sproc;
  unassign(`_62`), assign(L = [exports,locals,globals,`options`,`description`], nu = "AVZ42_61");
  assign(omega=interface(warnlevel), cs = ((k::integer, s::symbol) -> [assign('d' = [op(k, eval(s))],
    assign('p' = [seq(convert(d[v], string), v = 1 .. nops(d))], assign('t' = [seq(convert(
    substring(p[v], searchtext(":-", p[v]) + 1 .. -1), symbol), v = 1 .. nops(p))], op(t))]);
  assign(N = () -> `if` (rhs(op(args)) = NULL, NULL, args)), `if` (nargs = 1, NULL,
  [WARNING("%1 %2 %3", whattype(eval(M)), M, `if` (Sproc(M, 'h') = false,
  op([has been activated in the current session`, assign(z = 14)]), cat(`is in library`,
  convert(h, name))), `if` (z <> 14, assign(args[2] = h), NULL)]);
if type(M, procedure) then RETURN(eval(ParProc(M)))
else
  assign(a = cat("_62` := ", convert(eval(M), string))), interface(warnlevel = 0);
  assign(b = map2(Search, a, [ `module`, `export`, `(` ])); assign('c' = cat(a[1 .. b[1][1] - 1],
  "proc", a[b[3][1] .. b[2][1] - 1], a[searchtext(";", a, b[2][1] .. -1) + b[2][1] .. b[1][1] - 1],
  "proc;")), writebytes(nu, c), close(nu);
  (proc) read nu; remove(nu) end proc(), assign('a' = ParProc(`_62`)),
  interface(warnlevel = omega);
  assign(R = map(N, [[L[1] = op(cs(1, M))], [L[2] = op(cs(3, M))], [L[3] = op(cs(6, `_62`))],
  [L[4] = op(3, eval(`_62`))], [L[5] = op(5, eval(`_62`))]])), unassign(`_62`),
  RETURN(array(1 .. nops(R), 1 .. 1, R))
end if
end proc
Sproc:= proc(P::symbol)
local k, L, R, h, s, omega;
  `if` (type(P, procedure) or type(P, `module`), assign(L = [libname], R = {},
  s = CF(cat(CDM(), "\\lib")), RETURN(false)), assign(omega = (proc(R, s)
  local a, b;
    assign(a = map(CF, R), b = `if` (type(P, package), package, `if` (type(P, procedure),
    Proc, Mod)));
    if {s} = a then b, Maple elif member(s, a) then b, `Maple&User` else b, User end if
  end proc));
  `if` (type(P, builtin), RETURN(true, `if` (1 < nargs, assign(args[2] =
  [Proc, builtin, Builtin(P), s]), 7)), 7);
  try assign(h = IO_proc(P)),
  `if` (type(h, integer), RETURN(true, `if` (1 < nargs, assign(args[2] = [Proc, iolib, h, s]), 7)), 7)
  catch "" : seq(`if` (search(convert(march(list, L[k]), string), cat("[", P, ".m", "]),
  assign('R' = {op(R), L[k]}), NULL), k = 1 .. nops(L))
end try;

```

```

`if (R = {}, RETURN(false), RETURN(true, `if (nargs = 2,
assign([args][2] = [omega(R, s), R]), NULL)))
end proc

```

The combined use of the procedures *ParProc1* and *Sproc* provides the more differentiable localization of the program objects in the *Maple* environment (*library and/or the current session*).

Typical examples of the procedure use:

```

> ParProc(MkDir, ParProc(came)); map(ParProc, ['add', march, goto, iostatus, seq]);
matrix([[Arguments = (F::{string, symbol})], [locals = (cd, r, k, h, z, K, L, Lambda, t, d,
omega, omega1, u, f, s, g, v)]], matrix([[Arguments = (E::anything)], [locals = (f, h)],
[globals = (__Art_Kr_)]]]
[builtin function, 138, iolib function, 31, builtin function, 182, iolib function, 13,
builtin function, 253]

```

```

> Sproc(goto, 's'), s;

```

```

true, [Proc, built-in, 193, "c:\program files\maple 9\lib"]

```

```

> Sproc(MkDir, 'h'), h;

```

```

true, [Proc, User, {"c:/program files/maple 9/lib/userlib"}]

```

```

> Sproc(RANS_IAN_RAC_REA, 'z'), z;

```

```

false, z

```

```

> Sproc(LinearAlgebra, 'p'), p;

```

```

true, [package, Maple, {"C:\Program Files\Maple 9/lib"}]

```

**MmF** - merge of files created by the `save` statement of the package

**mmf**

Call format of the procedures:

**MmF(F, F1 {, h})**

**mmf(F, G, H)**

Formal arguments of the procedures:

**F, F1, G** - symbols or strings defining names or full paths to the merged datafiles

**H** - symbol or string defining name or full path to a target *m*-file

**h** - (optional) an arbitrary valid *Maple* expression

Description of the procedures:

The successful procedure call **MmF(F1, F2)** returns the *NULL* value, i.e. nothing, providing merge of datafiles (within definitions of program modules and procedures composing their) specified by their qualifiers **F1** and **F2**, and which represent the result of saving of the program objects by means of the built-in function `save`. Such merge result updates the first file **F1** with preservation of its format. In addition, the files of different formats can be united, namely: the input *Maple* language format or internal *Maple* format.

The procedure call **MmF(F1, F2, h)** with the optional third argument **h** provides removing out of a current session of those objects, which have been activated as a result of merge of datafiles **F1** and **F2**, and which earlier either have not been made active, or do not belong to libraries logically linked with the main *Maple* library. The procedure has a series of rather useful appendices. In particular, on the basis of the **MmF** procedure an algorithm of converting of datafiles of the input *Maple* language format into the internal *Maple* format is being enough easily implemented.

In addition, it is necessary to have in mind an important circumstance linked with impossibility of correct saving of definitions of program modules in *m*-files (*the internal Maple format*) by means of the `save` function. Therefore, if the merged datafiles contain program modules in the internal *Maple* format which were saved by means of the function `save`, then and the *MmF* procedure will incorrectly fulfill the merging. With this purpose, the procedure outputs the appropriate warning. The procedure *MmF* handles the basic erratic and especial situations.

At last, the *mmf* procedure is intended for merge of two compatible *m*-files with saving of the result in a new *m*-file. The procedure call *mmf(F, G, H)* merges two *m*-files *F* and *G* compatible relative to the *Maple* releases with return of full path to a target *m*-file *H* and with output of an appropriate warning. If a target *m*-file does not exist, it will be created. The merged *m*-files must have different names or full paths to them and must be created by the compatible *Maple* releases, otherwise errors arise. The procedure call handles all basic erratic and especial situations and does not affect the state of the current *Maple* session. The procedure has a series of rather useful appendices. In particular, on the basis of the *mmf* procedure the procedure *save3* has been implemented which correctly saves program modules in datafiles of the internal *Maple* format.

```

MmF := proc(F1::file, F2::file)
local act, a, b, c, d, k, Art, Kr, omega;
  omega := proc(f)
    local nu;
    assign(nu = interface(warnlevel)), interface(warnlevel = 0);
    try parse(readbytes(f, 'TEXT', infinity), 'statement')
    catch "incorrect syntax in parse:": close(f); interface(warnlevel = nu); return false
  end try;
  close(f), interface(warnlevel = nu), true
end proc;
if map(Ftype, {F1, F2}) = {".m"} or Ftype(F1) = ".m" and omega(F2) or Ftype(F2) = ".m"
  or map(omega, {F1, F2}) = {true} then assign(act = {anames('procedure'), anames('module')}),
  Art = [], Kr = [] else error "one or both files %1 are not readable", {F1, F2} end if;
if Rmf(F1) and Rmf(F2) then assign(a = [Read1([F1], 'b'), b, Read1([F2], 'c'), c])
  else error "the merged m-files should correspond to the current Maple release" end if;
  d := {seq(op([op(a[k][ 'procs' ]), op(a[k][ 'mods1' ]), op(a[k][ 'mods2' ])]), k = [1, 2])};
  seq('if (type(eval(d[k]), 'procedure'), assign('Art' = [op(Art), d[k]]),
  'if (type(eval(d[k]), 'module`), assign('Kr' = [op(Kr), d[k]], 1)), k = 1 .. nops(d)),
  'if (Kr <> [], WARNING("uncorrect saving of modules is highly probable!", 6);
  'if (cat(" ", F1)[-2 .. -1] <> ".m" or Kr = [], (proc() save args, F1 end proc)(op(d)),
  (proc() modproc(op(Kr)); save args, F1 end proc)(op(Art), op(Kr)));
if 1 < nargs then for k in d do
  if not (member(k, act) or ismLib(k)) then unprotect(k); unassign(k); next end if end do
end if
end proc
mmf := proc(F::file, G::file, H::{string, symbol})
local a, b, f;
  if not map(IsFtype, {F, G}, ".m") = {true} then error "at least one file has type different
  from '.m'" end if;
  if CF(F) = CF(G) then error "files are identical" else assign(a = map('readbytes', [F, G],
  'TEXT', 2)) end if; close(F, G);

```

```

if not SD(a[1][2]) = SD(a[2][2]) then error "m-files inhere in incompatible releases" end if;
`if (cat(" ", H)[-2 .. -1] <> ".m", assign(f = cat(H, ".m")), assign(f = H)),
    assign('a' = interface(warnlevel));
if not type(f, 'file') then interface(warnlevel = 0); Mkdir(f, 1); interface(warnlevel = a) end if;
assign('a' = readbytes(F, infinity), 'b' = readbytes(G, infinity)), close(F, G);
writebytes(f, [op(a[1 .. -2]), 44, 10, op(b[6 .. -1])]), close(f); WARNING("merge result of two
compatible files of the internal Maple format is in file <%1>", f), f
end proc

```

Typical examples of the procedure use:

```

> MM:=module() export x,m; m:=module() export y; y:=() -> `(args) end module end
module: MM1:=module() local F,m; export a; option package; m:=module() export y;
y:=() -> `(args) end module; F:=() -> m:- y(args); a:=F end module: module SVEGAL()
export x; x:=() -> `(args)/nargs^6 end module: PP:=proc() `(args)/nargs^2 end proc:
PP1:=proc() `(args)/nargs end proc: PP2:=proc() `(args)/nargs end proc: module GS ()
export x; x:=() -> `(args)/nargs^3 end module: save(MM, PP, "C:/RANS/Temp/File1");
save(GS, PP1, "C:/RANS/Temp/File1.m"); Read1(["C:/RANS/Temp/File1", 'h'), eval(h);
    table([mods2 = {}, mods1 = {MM}, vars = {}, procs = {PP}])
> Read1(["C:/RANS/Temp/File1.m", 'h'), eval(h);
    table([mods2 = {GS}, mods1 = {}, vars = {}, procs = {PP1}])
> MmF("C:/RANS/Temp/File1", "C:/RANS/Temp/File1.m");
    Warning, uncorrect saving of modules is highly probable!
> Read1(["C:/RANS/Temp/File1", 'h'), eval(h);
    Error, (in unknown) attempting to assign to `GS` which is protected
    table([mods2 = {GS}, mods1 = {MM}, vars = {}, procs = {PP1, PP}])
> map(type, [PP, PP1], procedure), map(type, [MM, GS], `module`);
    [true, true], [true, true]
> restart: MmF("C:/RANS/Temp/File1", "C:/RANS/Temp/File1.m", 61);
    Error, (in unknown) attempting to assign to `GS` which is protected
    Warning, uncorrect saving of modules is highly probable!
> map(type, [PP, PP1], procedure), map(type, [MM, GS], `module`);
    [true, false], [false, true]
> F:= "C:/rans/academy/vilnius\\Res.m": G:= "C:/rans/academy/vilnius\\Res1.m":
> mmf(F, G, "C:/RANS/IAN\\Academy/Grodno\\grsu");
    Warning, merge result of two compatible files of the internal Maple format is in file
    <C:/RANS/IAN\Academy/Grodno\grsu.m>
    "C:/RANS/IAN\Academy/Grodno\grsu.m"
> read("C:/RANS/IAN\\Academy/Grodno\\grsu.m");
> mmf(F, F, "C:/RANS/IAN\\Academy/Grodno\\grsu");
    Error, (in mmf) files are identical
> mmf(F, "C:/temp/aaa.m", "C:/RANS/IAN\\Academy/Grodno\\grsu");
    Error, (in mmf) m-files inhere in incompatible releases
> mmf("C:/temp/aaa.m", "C:/archive/abs.txt", "C:/RANS/IAN\\Academy/Grodno\\grsu");
    Error, (in mmf) at least one file has type different from '.m'

```

**mod21** - converting of a program module of the second type into the first type

Call format of the procedure:

**mod21**(M {, h})

Formal arguments of the procedure:

**M** - symbol defining a program module of arbitrary type

**h** - (*optional*) symbol or string defining a directory

Description of the procedure:

A series of inconveniences of operation with program modules of the second type, which in a series of cases require the more complex algorithms of their handling at software making, bring up a problem of program converting them into modules of the first type. In addition, a program module of the *first* type is understood as the module named by construction of view "**Name:= module () ...**", whereas a program module of the *second* type is characterized by determinative construction of view "**module Name () ...**". Some useful considerations on use of the program modules of the second type can be found in our books [12, 29-33,43]. Rather simple, but useful enough procedure *mod21* solves the given problem.

The procedure call *mod21(M)* returns the *NULL* value, i.e. nothing, providing a converting in a current session of a program module of the second type specified by the actual argument **M** into the equivalent program module of the first type of the same name. If the procedure call *mod21(M, h)* uses the second optional argument **h**, the argument is considered as a directory in which should be saved a datafile "M.mod1" with definition of the converted program module **M** (if the **h** directory is absent, then it will be created with arbitrary nesting level). In this case, the procedure call returns full path to a datafile with the saved converted program module **M**. A datafile with the saved program module has the input *Maple* language format. The examples below illustrate the said.

```

mod21 := proc(M::`module`)
local a, b, c, d, t, p, k, v, sf, h, nu;
assign(a:=convert(eval(M), 'string'), t={}, p=[], sf = ((x, y) -> `if`(length(y) <= length(x), true, false)));
  assign(b = Search2(a, {"module "}), assign('b' = [seq(k + 5, k = b)]);
  assign(c = {seq(nexts(a, b[k], "() ", k = 1 .. nops(b))});
  seq(assign('p' = [op(p), a[c[k][1] + 1 .. c[k][2] - 1]]), k = 1 .. nops(c)), assign('p' = sort(p, sf));
  p := [seq(op([assign('r' = Search2(p[k], {":-"})), `if`(r <> [] and p[k] <> ":-", p[k][2 .. r[-1] + 1],
    `if`(p[k] = " ", NULL, cat(p[k][1 .. -2], ":-")))), k = 1 .. nops(p))]; p := [seq(k = "", k = p)];
  seq(`if`(2 < c[k][2] - c[k][1], assign('t' = {op(t), v $ (v = c[k][1] + 1 .. c[k][2] - 1)),
    NULL), k = 1 .. nops(c));
eval(parse(cat("unprotect(", M, ")", assign("", M, "="), SUB_S(p, dsps(a, t), ""))));
if nargs = 2 then
  if type(args[2], 'dir') then h := cat(args[2], "\\ ", M, ".mod1"); save M, h; h else
  assign(nu = interface(warnlevel)), interface(warnlevel = 0),
  assign('h' = cat(MkDir(args[2]), "\\ ", M, ".mod1"));
  (proc() interface(warnlevel = nu); save M, h end proc())(), h;
  WARNING("module <%1> has been converted into the first type, and saved in datafile
    <%2>", M, h)
  end if
end if
end proc

```

Typical examples of the procedure use:

```

> module SVEGAL() export x,y; x:=() -> `+`(args)/nargs; y:=() -> [args, nargs] end module;
> type(SVEGAL, mod1), mod21(SVEGAL), type(SVEGAL, mod1), SVEGAL:- x(61,56,36);
                                     false, true, 51

```

```
> module Art_Kr () export SR, DS; SR:=() -> `+(args)/nargs; DS:=() -> sqrt(sum((args[k] -
SR(args))^2, k=1..nargs)/(nargs-1)) end module: type(Art_Kr, mod1), mod21(Art_Kr,
"C:/RANS/IAN/RAC/REA"), type(Art_Kr, mod1);
false, "C:/RANS/IAN/RAC/REA\Art_Kr.mod1", true
```

**islabel** - check of presence of goto-statements in procedures

Call format of the procedure:

**islabel(P::procedure {, T::symbol})**

Formal arguments of the procedure:

**P** - a procedure to be tested

**T** - (optional) an assignable name

Description of the procedure:

The procedure call **islabel(P)** returns the *true* value, if a procedure **P** uses *goto*-statements; otherwise, the *false* value is returned. While the procedure call **islabel(P, T)** through argument **T** additionally returns the set of names of all labels conditioned by *goto*-statements in the procedure **P**. In addition, the procedure checks the admissibility of labels used in the procedure **P**. At detection of such situation, the procedure outputs appropriate warning. The global nature of labels causes similar situations. For labels robustness of a procedure can be used the procedure *Lprot* that allows to assign attribute "protected" to the procedure labels.

```
islabel := proc(P::procedure)
```

```
local a, b, c, d, k, p, h, t;
```

```
assign(d = {}, a = interface(verboseproc), interface(verboseproc = 3);
```

```
assign(b = convert(eval(P), 'string'), p = {}), assign(c = Search2(b, {" goto(")},
interface(verboseproc = a));
```

```
if c = [] then false else
```

```
if search(b[1 .. c[1]], "unassign(", 't') then h := {parse(b[t + 8 .. nexts(b, t, ")])[2]]} end if;
```

```
for k in c do d := {op(d), cat(`, b[k + 6 .. nexts(b, k, ")])[2] - 1]} end do;
```

```
true, seq(`if (type(eval(d[k]), 'symbol'), NULL, assign('p' = {op(p), d[k]})), k = 1 .. nops(d)),
```

```
`if (p = {}, NULL, `if (map(eval, map(eval, h)) = map(eval, d), NULL,
```

```
WARNING("procedure <%1> contains invalid labels %2; error is possible
```

```
at the procedure call", P, `if (type(h, 'symbol'), p, p minus h)))),
```

```
`if (1 < nargs and type(args[2], 'symbol'), assign(args[2] = d), NULL)
```

```
end if
```

```
end proc
```

Typical examples of the procedure use:

```
> islabel(MkDir); => false
```

```
> islabel(mwsname, R), R; => true, {VGS}
```

```
> islabel(sin, T), T; => false, T
```

```
> islabel(mapleacs); => false
```

```
> islabel(minmax3d, T), T; => true, {Cycle}
```

```
> type(islabel, boolproc); => true
```

```
> islabel(A1, h), h; => true, {56, A, Fin}
```

```
> B:= 63: A1:= proc() if `+(args)>10 then goto(A) else goto(B) end if;
```

```
A; lprint(args); goto(Fin);
B; lprint(args[1]);
Fin; NULL
end proc:
> islabel(A1, h), h; ⇒ true, {63, A, Fin}
Warning, procedure <A1> contains invalid labels; error is possible at the procedure call
> A1(1, 2);
Error, (in A1) goto to an undefined or unreachable label
> A1(1, 2, 63); ⇒ 1, 2, 63
> B:=63: A1:= proc() unassign('A','B','Fin'); if `+(args)>10 then goto(A) else goto(B) end if;
A; lprint(args); goto(Fin);
B; lprint(args[1]);
Fin; NULL
end proc:
> islabel(A1, h), h; ⇒ true, {63, A, Fin}
> A1(1, 2); ⇒ 1
> A1(1, 2, 63); ⇒ 1, 2, 63
```

#### **mod\_proc** - converting of a program module into procedure

**ModProc**

**modproc**

Call format of the procedures:

**mod\_proc**(M {, F})

**ModProc**(M)

**modproc**(args {, `\$`})

Formal arguments of the procedures:

**M** - symbol defining name of a program module

**F** - (*optional*) symbol or string defining a filename or full path to it

**args** - a sequence of program modules names

**`\$`** - (*optional*) key symbol defining a saving mode of result

Description of the procedures:

The package has a rather developed mechanism of profiling of procedures - of main program objects of the *Maple* language. Between that, for program modules the given method of a profiling in its direct view is inapplicable. In our books [28-30, 32, 33, 41, 240] we have offered other profiling method, whose essence consists in replacement of a program module by the equivalent procedure. For automation of the given process, it is possible to use two rather useful procedures **mod\_proc** and **ModProc**.

With the purpose of the equivalent transformation of a program module (*whose definition has been evaluated in the current session*) into the procedure, the **mod\_proc** procedure can be substantially useful. This procedure returns a procedure with name **proc\_N** which is created on the basis of the given program module, where **N** is name of a source module named by any permissible way.

The **mod\_proc**(M, F) procedure admits up to two actual arguments. In addition, the first obligatory actual **M** argument specifies name of a program module whose definition has been evaluated in the current session. In this case the procedure call **mod\_proc**(M) returns the procedure, which is fully equivalent to the initial program module **M**. Whereas the procedure call with two actual arguments along with the above result saves the generated procedure in a file, the full path to which or its name is specified by the second actual **F** argument.

Because of the above-mentioned converting of a program module into the procedure the exported variables of the module, become the local variables of the procedure, which are being returned during its execution. In addition, the call of the created procedure without arguments returns a sequence of local variables (*exported variables of the source module*). The access to these variables is provided by a way visually represented by examples below. In particular, the **mod\_proc** procedure seems as a rather useful facility for a deriving and saving of the source texts of the package modules.

The **ModProc(M)** procedure uses only one actual **M** argument - name of an evaluated program module, returning the equivalent procedure. Single restriction for the procedure is the necessity of absence for the converted module **M** of the nested modules. In a case of detection of such module, the error arises. Successful execution of the procedure returns the procedure with name of a view **mp\_M** and with output of the appropriate warning. The exported variables of the program module are being put into *global* declaration of the external procedure - of the result of converting of the **M** module. The procedure created on the basis of a program module can be profiled by the standard method. The call **mp\_M()** provides access to variables exported by an initial **M** module.

In our books [28-30, 32, 33, 41, 240] the question of incorrect saving of program modules in files of the internal *Maple* format (*m-files*) has been considered in detail enough. We have offered a method of a converting of program modules into equivalent procedures, which next are correctly saved in the files of both formats by means of the standard function *'save'* of the package. This method is based on the **modproc** procedure represented below.

As the actual arguments the **modproc** procedure admits a sequence of names of program modules and, possibly, key symbol *'\$'*, which defines a saving mode of the result in files of both formats. The procedure call

**modproc(M1, M2, ..., ..., '\$', ..., Mk) (p=1..k)**

returns the *NULL* value, i.e. nothing, providing in the current session the converting of **Mp** program modules into **Mp** procedures which are correctly saved by means of the *'save'* function in files of both *Maple* formats. At presence among the actual arguments of key symbol *'\$'*, the converted **Mp** procedures are saved in the current directory in files of both formats with names *'\_\$\$\$'* and *'\_\$\$\$m'* accordingly. In addition, the program modules of the second type are converted into modules of the first type without attribute *'protected'*. Because of execution, the procedure can output the appropriate warnings.

Syntax of the first access to the exported variables of such converted program module differs a little from the standard and has the following view:

**Name(): Name:- exportX**

where *Name* - name of a program module and *exportX* is any variable exported by it. All subsequent accesses to such module in the current session use syntax adopted in *Maple*. The above said also concerns use of program modules, saved thus in the *m-files*, after reading of an *m-file* containing them, by means of the *'read'* function. The **modproc** procedure is a rather useful facility at making a various sort of software whose algorithm supposes a saving of program modules in files of the internal *Maple* format (*m-files*). The examples represented below enough lucidly illustrate both the results of usage of the above procedures, and algorithms used by them.

```
mod_proc := proc(M:='module')
local alpha, beta, m, n, s, t, f, f1, h, loc, ex, e, T, R, z, v, w;
  e:= (a, b, c) -> op([assign('h' = searchtext(c, a, b .. length(a)) + b - 1), h]);
  assign(s = cat(M, " := module "), f = cat([libname][1][1 .. 2], "\\ $Sveta"));
  T:= (proc(x) eval(M); save M, x; readbytes(x, 'TEXT', infinity) end proc)(f);
```

```

close(f), assign(R = map2(Search, T, ["module", "local", "export"]));
m := cat("proc_", T[searchtext(s, T) .. R[1][1] - 1], "proc() ");
assign(z = cat(M, ":-"), n = "", ex = T[R[3][1] + 7 .. e(T, R[3][1], ";")),
  `if (search(T[1 .. R[3][1]], "local", 't'), assign('m' = cat(m, T[R[2][1] .. e(T, R[2][1], ";") - 1,
  ",", ex)), assign('m' = cat(m, "local ", ex))), assign('m' = cat(m, T[e(T, R[3][1], ";") + 1 ..
  R[1][1] - 5], ex, " end proc;"));
seq(assign('n' = cat(n, `if (m[k] = " " and m[k + 1 .. k + 2] = ":-", "", m[k])), k = 1 .. length(m));
assign(loc = Search(n, z)), `if (loc = [], assign(v = n), [assign(alpha = n[1 .. loc[1] - 1],
  beta = n[loc[1] + length(z) .. length(n)], assign(v = cat(alpha, seq(n[loc[k] + length(z) ..
  loc[k + 1] - 1], k = 1 .. nops(loc) - 1), beta))]);
(proc(x, y) (proc(z, q) writebytes(q, z), close(q); read q end proc)(x, y),
  `if (search(y, "\\ $sveta"), fremove(y), fremove(f)), eval(cat(proc_, M))
end proc(v, `if (nargs = 2, args[2], f))
end proc
ModProc:= proc(M::`module`)
local G, N, T, z, t, W, h, F;
  assign(F = cat([libname][1][1 .. 2], `/$`)), (proc(x) save M, x end proc)(F);
W:= proc(S, K)
  local a, b, c, p, h;
  [assign('a' = [], 'b' = length(S), 'c' = length(K), 'p' = 1), seq(`if (search(S[p .. b], K, 'h'),
  assign('a' = [op(a), p + h - 1], 'p' = h + p), RETURN(a)), k = 1 .. length(S))]
end proc;
assign(T = readbytes(F, 'TEXT', infinity)), close(F), assign(N = W(T, "module")), unassign('t');
if 2 < nops(N) then error "module <%1> contains nested modules", M
else assign(h = cat('mp_', M)) end if;
`if (search(T, cat("unprotect(", M, "));"), assign('T' = T[length(M) + 16 .. N[2] + 6]), NULL);
N := map(searchtext, ["module", "export", "global", "end module"], T);
`if (N[2] = 0, WARNING("module <%1> has not of the exported names", M), NULL);
assign(G = cat("mp_", T[1 .. N[1] - 1], "proc")), assign(z = length(T));
`if (N[2] = 0, assign('G' = cat(G, T[searchtext(")", T) .. N[4] + 3], "proc;")),
  [search(T[N[2] .. length(T)], "proc", 'z'), `if (search(T[N[2] .. z + N[2] - 1], "global", 't'),
  assign('G' = cat(G, T[searchtext(")", T) .. N[2] - 1], "global", T[N[2] + 6 ..
  searchtext(";", T[N[2] .. N[2] + t - 1]) + N[2] - 2], "", T[N[2] + t + 6 .. N[4] + 3], "proc;")),
  assign('G' = cat(G, T[searchtext(")", T) .. N[2] - 1], "global", T[N[2] + 6 .. N[4] + 3],
  "proc;"))], writeline(F, G), close(F), (proc(y) read y end proc)(F), fremove(F);
eval(h), WARNING("module <%1> has been converted into procedure with name <%2>", M, h)
end proc
modproc:= proc()
local k, h, t, M, gs;
  assign(gs = (proc(s, n)
  local h;
  `if (search(s, cat(" ", n, " ()", 'h'), cat(s[1 .. h - 1], s[h + length(n) + 1 .. -1]), s) end proc)),
  assign(M = []), `if (nargs = 0, RETURN(NULL), seq(`if (args[k] = `$`, NULL, `if`
  type(eval(args[k]), `module`), [assign('M' = [op(M), args[k]], unprotect(args[k])),
  WARNING("<%1> is not a program module", args[k]))], k = 1 .. nargs));

```

```

if M = [] then error "converting has not been done - program modules are absent" end if;
for k to nops(M) do
  assign('h' = gs(convert(eval(M[k]), 'string'), M[k])), assign('h' = cat("", M[k],
    " := proc() RETURN(assign(" M[k], " = ", h, ")) end proc;")), writebytes("_$#@61", h),
    close("_$#@61");
  (proc) read "_$#@61"; remove("_$#@61") end proc())
end do;
`if`(member(`$`, {args}), null([(proc) save args, `$$$`; save args, `$$$m` end proc](op(M)),
  WARNING("the converted program modules have been saved in files <%1> and <%2>",
  cat(CDM(), `/_$$$`), cat(CDM(), `/_$$$m`))), NULL)
end proc

```

Typical examples of the procedure use:

```

> module M1() local a,b,c,A; global d; option package; description "RANS_IAN"; export
  sr, Dis; A:= module() export h; h:=() -> `+`(args)/nargs end module; sr:=() -> A:- h(args);
  Dis:=() -> add((args[k]-sr(args))^2, k=1..nargs)/(nargs-1); end module;
module M1 () local a, b, c, A; export sr, Dis; global d; option package; description "RANS_IAN";
  end module

```

```

> mod_proc(M1);

```

```

proc()
local a, b, c, A, sr, Dis;
global d;
option package;
description "RANS_IAN";
A := module() export h; h := () -> `+`(args)/nargs end module;
sr := () -> A:- h(args);
Dis := () -> add((args[k] - sr(args))^2, k = 1 .. nargs)/(nargs - 1);
sr, Dis
end proc

```

```

> 30*seq(proc_M1()[k](42,47,67,62,89,96), k=1..2);

```

```

2015, 14249

```

```

> mod_proc(M1, "C:/RANS/Temp/Art.txt"); restart; read("C:/RANS/Temp/Art.txt");

```

```

proc()
local a, b, c, A, sr, Dis;
global d;
option package;
description "RANS_IAN";
A := module() export h; h := () -> `+`(args)/nargs end module;
sr := () -> A:- h(args);
Dis := () -> add((args[k] - sr(args))^2, k = 1 .. nargs)/(nargs - 1);
sr, Dis
end proc
proc_M1 := proc()
local a, b, c, A, sr, Dis;
global d;
option package;

```

```

description "RANS_IAN";
  A := module() export h; h := () -> `+(args)/nargs end module;
  sr := () -> A:- h(args);
  Dis := () -> add((args[k] - sr(args))^2, k = 1 .. nargs)/(nargs - 1);
  sr, Dis
end proc
> 6*proc_M1()[1](42,47,67,62,89,96), 30*proc_M1()[2](42,47,67,62,89,96);
403, 14249
> Kr:= module() local a; global c; export x,y,z; assign(x=() -> `+(args)/nargs), z=()-> [args,
nargs]); y:= proc() local k; c*sum(args[k]^2, k=1..nargs)/x(args) end proc end module:
ModProc(Kr);
Warning, module <Kr> has been converted into procedure with name <mp_Kr>
proc()
local a;
global x, y, z, c;
x := () -> `+(args)/nargs;
y := proc() local k; c*sum(args[k]^2, k = 1 .. nargs)/x(args) end proc;
z := () -> [args, nargs]
end proc
> c:=2003: mp_Kr(): 3*x(61,56,36,40,7,14), 103*y(42,61), z(45,67,78);
107, 21972910, [45, 67, 78, 3]
> module M1() local a,b,c; global d; option package; description "RANS_IAN"; export sr,
Dis,A; A:= module() export h; h:=() -> `+(args)/nargs end module; sr:=() -> A:- h(args);
Dis:=() -> add((args[k]-sr(args))^2,k=1..nargs)/(nargs-1); end module: Kr:=module() local
a; global c; export x,y,z; assign(x=()->`+(args)/nargs), z=()->[args,nargs]); y:=proc() local
k; c*sum(args[k]^2, k=1..nargs)/x(args) end proc end module: modproc(Kr, `$`,M1);
Warning, the converted program modules have been saved in files <C:\Program Files\Maple
9/_$$$> and <C:\Program Files\Maple 9/_$$$m>
> M1(): 6*M1:- sr(61,56,36,40,7,14), 30*M1:- Dis(61,56,36,40,7,14);
214, 14192
> restart: read(`_$$$m`): M1(): 6*M1:- sr(61,56,36,40,7,14), 30*M1:- Dis(61,56,36,40,7,14);
214, 14192
> c:=107: Kr(): 6*Kr:- x(61,56,36,40,7,14), 71*Kr:- y(61,56,36,40,7,14);
214, 2129574

```

In particular, the `mod_proc` procedure seems as a rather useful tool for receiving and saving the source codes of package modules. For example, by means of constructions of the following general view:

```

with(pacman): interface(verboseproc=3): mod_proc(pacman); 6 - 8
with(PackageManagement): interface(verboseproc=3): mod_proc(PackageManagement); 9/9.5

```

the specified information can be received for the package modules providing a set of utilities for interactive management by modules for releases 6-8 and 10 of *Maple*.

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

# Chapter 3.

## Software for work with symbols, strings, lists, sets and tables

In the given chapter, the tools extending possibilities of *Maple* of releases 6-10 at operation with expressions of type  $\{string, symbol, list, set, table\}$  are considered. These tools provide a number of useful procedures such as special kinds of converting; comparison of strings or/and symbols; case sensitive pattern searching; exhaustive substitutions into strings or symbols; inversion of symbols, strings or lists; reducing of multiplicity of entries of a symbol into a string; identification of entries of special symbols into a string; and others. The list structures play an extremely important role, defining the ordered sequences of elements. Since the *sixth* release, a possibility of substantial extending of operations with the list structures had appeared. As an example having the interesting practical appendices, we consider definition of algebra on a set of all lists having the same length. Algebraic operations the corresponding procedures, presented below, provide. A series of procedures of the section supports useful kinds of processing such as: a special converting of lists into sets, and vice versa; operation with rarefied lists; dynamic assignment of values to elements of a list or a set; evaluation of indices of a table over its entry; representation of a special type of tables; special kinds of exhaustive substitutions into lists or sets; a series of important kinds of sorting of nested lists, and also many others. In a series of cases, these tools simplify operation with *Maple* objects of type  $\{string, symbol, list, set, table\}$  in the *Maple* environment. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**nexts** – search of pattern **B**, the nearest to a pattern **A**

Call format of the procedure:

**nexts**(**S**:: $\{string, symbol\}$ , **A**:: $\{posint, string, symbol\}$ , **B**:: $\{string, symbol\}$  [, **r**::*anything*])

Formal arguments of the procedure:

- S** – symbol or string in which search is done
- A** – symbol or string defining the first pattern, or its position in **A**
- B** – symbol or string defining the second pattern
- r** – (*optional*) arbitrary expression defining the search mode

Description of the procedure:

Procedure **nexts**(**S**, **A**, **B** [, **r**]) provides search in string **S** of patterns **B**, the nearest to patterns **A** to the right or to the left. The procedure call with three arguments defines search to the right of **A** whereas call with four or more arguments defines search to the left of **A**. Procedure call **nexts**(**S**, **A**, **B** [, **r**]) returns the nested list, whose elements are 2-element lists (*if nested list contains one element only, the ordinary list is returned*). The first element of such sublist defines a position of pattern **A** in **S** whereas the second defines a position of pattern **B**, the nearest to **A** to the right or to the left accordingly. In addition, if the nearest to **A** pattern **B** has not been found, the returned list will

contain position of pattern **A** only. Moreover, the argument **A** may be defined as a pattern, and position of a separate symbol in **S**. If the second argument **A** is absent in **S**, the procedure call returns *false*. If any actual argument is empty, the procedure call causes erroneous situation.

```

nexts := proc(S::{string, symbol}, A::{posint, string, symbol}, B::{string, symbol})
local a, b, s, k, t, n;
  if map(member, {S, B, A}, {'', ''}) = {'false'} then
    try
      assign(s = cat('', S), b = [], n = `if` (type(A, 'posint'), 1, length(A)));
      `if` (type(A, {'string', 'symbol'}), assign(a = Search2(s, {A})),
        assign(`if` (A <= length(s), assign(a = [A]),
          proc(s) error "2nd argument must be <= %1", length(s) end proc(s)))
      catch: error "wrong type of arguments in nexts call - %1", [args]
    end try
  else error "arguments cannot be empty", [S, A, B]
end if;
if a = [] then `false`
elif nargs = 3 then
  for k in a do `if` (search(s[k + 1 + n .. -1], B, 't'), assign('b' = [op(b), [k, t + k + n]]),
    assign('b' = [op(b), [k]]))
  end do;
  `if` (nops(b) = 1, op(b), b)
else
  for k in a do
    assign('a' = Search2(s[1 .. k - 1], {B})), `if` (a = [], assign('b' = [op(b), [k]]),
    assign('b' = [op(b), [k, a[-1]]]))
  end do;
  `if` (nops(b) = 1, op(b), b)
end if
end proc

```

Typical examples of the procedure use:

```

> S:="aaacccacaaaccaaacaacdddccrtdrtbbaaabaaabaaahyrebaaa":
> nexts(S, 29, b), nexts(S, 29, b, 8); => [29, 33], [29]
> nexts(S, '', b); => Error, (in nexts) arguments cannot be empty,
[aaacccacaaaccaaacaacdddccrtdrtbbaaabaaabaaahyrebaaa, , b]
> nexts(S, aaa, b); nexts(S, aaa, bb); => [[1, 33], [10, 33], [15, 33], [35, 42], [39, 51], [43, 51], [52]]
[[1, 33], [10, 33], [15, 33], [35], [39], [43], [52]]
> nexts(S, aaa, ba); => [[1, 34], [10, 34], [15, 34], [35, 42], [39, 51], [43, 51], [52]]
> nexts('', aaa, ba); nexts(S, aaa, bd); => Error, (in nexts) arguments cannot be empty, [, aaa, ba]
[[1], [10], [15], [35], [39], [43], [52]]
> nexts('', aa, ba), nexts(S, aaa, ``); => Error, (in nexts) arguments cannot be empty, [, aa, ba]
> nexts(S, xyz, ba); => false
> nexts(S, "cr", rt); => [27, 31]
> nexts(S, aaa, b, 1); nexts(S, aaa, bb, 2); => [[1], [10], [15], [35, 34], [39, 38], [43, 42], [52, 51]]
[[1], [10], [15], [35, 33], [39, 33], [43, 33], [52, 33]]
> nexts(S, ccc, t, 1), nexts(S, crt, bb, 2), nexts(S, crt, bb); => [[4], [5]], [27], [27, 33]

```

\_SL - dynamic assignment of values to elements of a list or set

Call format of the procedure:

\_SL(L, expr {, p})

Formal arguments of the procedure:

**L** - a list or a set

**expr** - an expression

**p** - a positive integer (*posint*)

Description of the procedure:

The procedure call **\_SL(L, expr)** returns the *NULL* value, providing the assignment of a value, given by the second actual **expr** argument, to all elements of a list or set defined by the first actual **L** argument; the replacement of elements of the initial **L** argument is being done in situ. In addition, assignments are done for the unassigned names only.

If the third optional actual **p** argument has been coded, the procedure call **\_SL(L, expr, p)** provides the assignment of **expr** value to the **p**th element of a list or set **L**. The given procedure allows easily to actualize a mechanism of dynamic assignment of values to variables both outside and in body of a procedure, what essentially facilitates programming of many problems of various purposes.

```
_SL := proc(L::{set, list}, a::anything)
local k;
  if(3 <= nargs, if(belong(args[3], 1 .. nops(L)), assign(if(type(op(args[3], L), 'symbol'),
op(args[3], L) = a, NULL)), proc() error "Invalid 3rd argument" end proc()),
assign(seq(if(type(eval(L[k]), 'symbol'), op(k, L) = a, NULL), k = 1 .. nops(L))))
end proc
```

Typical examples of the procedure use:

```
> restart; _SL([Art, Kr, V, G, Sv, Arn], 2003); [Art, Kr, V, G, Sv, Arn];
[2003, 2003, 2003, 2003, 2003, 2003]
> Art, Kr:= 14, 7: _SL([Art, Kr, x, y, z, h], 1942), [Art, Kr, x, y, z, h];
[14, 7, 1942, 1942, 1942, 1942]
> k:=61: L:=[y1, y2, y3, y4,y5,y6,y7]: seq(_SL(L, k^k, k),k=1..nops(L)); k, eval([seq(cat(`, y, k),
k=1..6))];
61, [19421, 19422, 19423, 19424, 19425, 19426]
> _SL([1942, 1947, X, 1967, 1962, Y, 1989, 1996, Z], 2003), [X, Y, Z];
[2003, 2003, 2003]
```

**mlsnest - evaluation of levels of nesting of lists and sets**

Call format of the procedure:

mlsnest(L::{list, set} {, 't'})

Formal arguments of the procedure:

**L** - a list or set

**t** - (*optional*) an assignable name

Description of the procedure:

In a series of cases in problems of operation with the nested lists and sets a task of definition of elements with maximum levels of nesting arises. In this connection the procedure **mlsnest** can be useful enough. The procedure call **mlsnest(L)** returns two-element nested integer list whose first element is list whose first element is number of element in the list (*set*) **L** and its second element defines minimal level of nesting, whereas the second list defines element of **L** with maximal level of



Moreover, if the `delel` procedure uses a word `{delel|delel1}` as the last actual argument, the procedure in addition provides two more modes of elements deletion out of the set/list `L`, namely: if the word `delel` had been coded, an element `L` is deleted if its number or value belong to set `{x, y, z, ...}`; if the word `delel1` had been coded, elements deletion is done at first over their numbers, and then on their values. Similar modes are entered in view of the circumstance that number of an element can coincide with its value. At last, the procedures calls `delel(L)`, `delel(L, 'delel')`, `delel(L, 'delel1')` return the argument `L`. Procedure `delel` appears enough useful at processing sets and lists.

```

delel := proc(L::{list, set})
local a, b, k;
if nargs = 1 or nargs = 2 and member(args[2], {'delel', 'delel1'}) then L
elif args[-1] = 'delel' then
    [seq('if' (member(L[k], {args[2..-2]}) or member(k, {args[2..-2]}), NULL, L[k]), k = 1 .. nops(L));
    convert(%, whattype(L))
elif args[-1] = 'delel1' then assign(a = {}, b = {}), seq('if' (type(k, 'posint'), assign('a' = {op(a), k}),
    assign('b' = {op(b), k})), k = [args[2 .. -2]]);
    [seq('if' (member(k, a), NULL, 'if' (member(L[k], b), NULL, L[k])), k = 1 .. nops(L));
    convert(%, whattype(L))
else try assign(a = [args[2 .. -1]]) catch "" : NULL end try;
if type(a, list('posint')) then [seq('if' (member(k, a), NULL, L[k]), k = 1 .. nops(L));
    convert(%, whattype(L))
else
    [seq('if' (member(L[k], a), NULL, L[k]), k = 1 .. nops(L)); convert(%, whattype(L))
end if
end if
end proc
    
```

Typical examples of the procedure use:

```

> L:= [a, b, 1, g, sin(x), a+b, 9, (a+b)/(c+d), a, x, y, 8, 16, svegal, tru, a]: delel(L), delel(L, 'delel');
    [a, b, 1, g, sin(x), a+b, 9, (a+b)/(c+d), a, x, y, 8, 16, svegal, tru, a],
    [a, b, 1, g, sin(x), a+b, 9, (a+b)/(c+d), a, x, y, 8, 16, svegal, tru, a]
> delel(L, 4, 1, 16, a, tru, a+b); => [b, g, sin(x), 9, (a+b)/(c+d), x, y, 8, svegal]
> delel(L, 4, 1, 8); => [b, 1, sin(x), a+b, 9, a, x, y, 8, 16, svegal, tru, a]
> delel(L, 4, a, sin(x), 5, (a+b)/(c+d), 20, 1, 6, 'delel'); => [b, 9, x, y, 8, 16, svegal, tru]
> delel(L, 4, a, sin(x), 5, (a+b)/(c+d), 20, 1, 6, 'delel1'); => [b, 1, 9, x, y, 8, 16, svegal, tru]
    
```

*sllj* - a transposition of the corresponding elements of sublists of listlist-list

Call format of the procedure:

`sllj(L::listlist, P:: set(posint) {, G::list({function, procedure}))`

Formal arguments of the procedure:

- L** - a list of listlist-type
- P** - a set of numbers of posint-type
- G** - (optional) a list of procedures and/or functions

Description of the procedure:

In a series of the problems dealing with the nested list structures, first of all of listlist-type, a problem of reorganization (*permutation*) of the corresponding elements of their sublists arises. In addition, under the corresponding elements of a nested list of listlist-type

$[[x_1, x_2, \dots, x_j, \dots, x_n], [y_1, y_2, \dots, y_j, \dots, y_n], [z_1, z_2, \dots, z_j, \dots, z_n], \dots, [h_1, h_2, \dots, h_j, \dots, h_n]]$

we shall understand a tuple of elements  $\langle x_j, y_j, z_j, \dots, h_j \rangle$ . By transposition we shall understand the operation over an initial list  $L$  of *listlist*-type as a result of which a new list of the same type is returned, but the corresponding elements in the given  $j$ -position of that have been reorganized according to some function/procedure from  $nops(L)$  variables, which return result of permutation of tuple  $\langle x_j, y_j, z_j, \dots, h_j \rangle$ . At absence of such procedure the inversion of elements of the specified tuple is supposed by default.

The procedure call  $sllj(L, P)$  returns the result of transposition of the corresponding elements of the nested list  $L$  of *listlist*-type, in the positions determined by elements of an integer set  $P$ . At absence of third optional argument  $G$  the transposition is reduced to inversion of the corresponding elements; otherwise, the transposition is defined by the functions/procedures from list  $G$  as it illustrates the fragment presented below. As the procedure managing the transposition of the corresponding elements of the *listlist*-list, procedure of the following simple form can act, for example:

$$F := (x_1, x_2, x_3, \dots, x_n) \rightarrow [T(x_1, x_2, x_3, \dots, x_n)]$$

where  $n = nops(L)$  and  $T$  - any function permuting  $x_j$ -arguments; in addition, in the list returned by  $F$ -function, variables can be duplicated

```

sllj := proc(L::listlist, P::set(posint))
local a, b, m, n, k, j, p;
if Empty(L) then error "1st argument is empty" elif Empty(P)
then error "2nd argument is empty" else assign(a=L, n= nops(L), m=nops(L[1]), p=nops(P))
end if;
if m < nops(P) or m < P[-1] then error "2nd argument is improper"
elif 2 < nargs then
  if not type(args[3], 'list'({'function', 'procedure'})) or nops(args[3]) <> p or
  {seq(type(args[3][k], 'arity'(n)), k = 1 .. p)} <> {`true`}
  then error "3rd argument is improper"
  end if
end if;
for j to nops(P) do b := [seq(a[k][P[j]], k = 1 .. n)];
  if 2 < nargs then b := eval(args[3][j](op(b))) else b := InvL(b) end if;
  seq(assign('a'[k][P[j]] = b[k], k = 1 .. n)
end do;
a
end proc

```

Typical examples of the procedure use:

```

> L:=[[1,2,3,4,5], [a,b,c,d,e], [x,y,z,r,t], [63,58,38,16,9], [v,g,s,k,a], [9,8,7,6,5], [0,0,0,0,0]]: sllj(L, {1,3,5});
[[0, 2, 0, 4, 0], [9, b, 7, d, 5], [v, y, s, r, a], [63, 58, 38, 16, 9], [x, g, z, k, t], [a, 8, c, 6, e], [1, 0, 3, 0, 5]]

```

```

> f:=proc(x1, x2, x3, x4, x5, x6, x7) [x1, x4, x3, x2, x5, x6, x7] end proc:

```

```

> f1:=proc(x1, x2, x3, x4, x5, x6, x7) [x7, x6, x5, x4, x3, x2, x1] end proc: sllj(L, {3, 5}, [f, f1]);

```

```

[[1, 2, 3, 4, 0], [a, b, 38, d, 5], [x, y, z, r, a], [63, 58, c, 16, 9], [v, g, s, k, t], [9, 8, 7, 6, e],[0, 0, 0, 0, 5]]

```

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

# Chapter 4.

## Software to support bit-by-bit processing of symbolic information

In this chapter, the tools, which support a *bit-by-bit* information processing in the *Maple* environment, are represented. The package does not possess tools of the similar type. The software offered by us is represented by six useful procedures such as **Bit**, **Bit1**, **xbyte**, **xbyte1**, **xNB** and **xpack**. These procedures serve for *bit-by-bit* information processing, i.e. the user has a possibility to operate with strings or symbols at a level of separate bits composing them.

**Bit** - *bit-by-bit information processing*

**Bit1**

**xbyte**

**xbyte1**

**xNB**

Call format of the procedures:

**Bit(B, bits)**

**Bit1(B1, bits)**

**xbyte(L)**

**xbyte1(L1)**

**xNB(N)**

Formal arguments of the procedures:

**B** - a symbol or a string of length **1**, or an integer **1**-element list

**B1** - an integer from the range **0..255**

**bits** - a leading variable of a polynomial

**L** - a list from **8** binary digits

**L1** - a positive binary number of length  $\leq 8$ , or a list from no more than **8** binary digits

**N** - a positive integer (*posint*)

Description of the procedures:

The tools represented below serve for a *bit-by-bit* information processing, i.e. the user has a possibility to operate with the symbols or strings on a level of separate bits composing them. The first **Bit** procedure allows effectively enough to execute *bit-by-bit* processing of separate symbols.

The procedure call **Bit(B, bits)** allows to execute the following basic *bit-by-bit* operations with a symbol, defined by the first actual **B** argument of the procedure (as argument can appear an one-element string or symbol of length **1**, and also 1-element list whose element defines a decimal code of a symbol from the range **0 .. 255**):

- (1) if the second actual **bits** argument has a view [**n1, n2, ...**], then the procedure call **Bit(B, bits)** returns the values of bits of symbol **B**, which are located in its **n<sub>j</sub>** positions (**n<sub>j</sub>** belong to the range **1..8**);

- (2) if the second actual *bits* argument has a view [ $n_1=b_1, n_2=b_2, \dots$ ], the decimal code of a symbol, obtained by replacement of values of bits of symbol **B** (which are in its positions  $n_k$ ) onto binary values  $b_k$ , is returned ( $n_k = 1 \dots 8$ ;  $b_k = \{0|1\}$ ).

- (3) if the second actual *bits* argument is a name of procedure defined over the lists, the result of application of the procedure to a list of values of all bits of a symbol, defined by the first actual **B** argument, is returned.

In addition, in two last cases the decimal code of a symbol - of the result of processing of the initial symbol, defined by the first actual **B** argument of the procedure, is returned.

The *Bit1(B1, bits)* procedure is a modification of the above *Bit* procedure. In the procedure the more effective algorithm of evaluations has been implemented, however a handling of special and erroneous situations are not fulfilled, and as the first actual **B1** argument only the decimal code of a symbol subjected to bit-by-bit processing is admitted. The second *bits* argument has remained the same. The given procedure is more reactive and can be successfully used in the cyclic computational constructions.

For providing of *bit-by-bit* symbolic processing two basic procedures *Bit* and *Bit1* represented above are intended. The simple procedure *xbyte(L)* fulfills an inverse function, returning a symbol with the given bit value defined by the actual **L** argument; as a value for actual **L** argument a list from 8 binary digits is used.

Provided that the arbitrary symbol "X" has a *bit-by-bit* representation **B::list(binary)**, for the procedures *xbyte* and *Bit* the following defining relations take place:

$$\text{"X"} = \text{xbyte}(\text{Bit}(\text{"X"}, [k\$k=1..8])) \quad \text{Bit}(\text{xbyte}(\text{B}), [k\$k=1..8]) = \text{B}$$

The procedure *xbyte1(L)* extends the above procedure *xbyte* onto positive binary numbers, i.e. the numbers consisting of numerals {0,1} only; for example **11001011**. In this case we can represent a symbol in the form of such binary number of length, not greater than 8; at a smaller length the procedure supplements a *bit-by-bit* representation at the left by zeroes. Moreover, a binary list of length < 8, obtained at the procedure call, is supplemented at the left by zeroes up to the standard list of length 8. Such approach allows in many cases more conveniently to represent the symbols at a level of bits composing them.

The simple procedure *xNB(N)* has been implemented by a one-string extracode and is useful enough at operation with information at the bit level. The procedure call *xNB(N)* returns the *true* value, if a positive integer **N** consists of binary numerals only, otherwise the *false* value is returned. The five procedures *Bit*, *Bit1*, *xbyte*, *xbyte1* and *xNB*, represented above, along with the introduced types {*digit*, *binary*, *byte*} create an quite satisfactory basis for operation with symbolic information at the *bit-by-bit* level. The examples represented below illustrate use of all procedures represented above for a *bit-by-bit* information processing in the *Maple* environment.

```
Bit := proc(B::string, symbol, list(integer)), bits::{procedure, list({integer, equation})})
local a, b, c, k;
c := (x) -> [seq(parse(x[k]), k = 1 .. length(x));
if type(B, {'symbol', 'string'}) and length(B) = 1 then
a := c(cat("", convert(op(convert(B, 'bytes')), 'binary'))); b := [0 $ ('k' = 1 .. 8 - nops(a)), op(a)]
elif type(B, 'list('integer')) and nops(B) = 1 and belong(B[1], 0 .. 255) then
a := c(cat("", convert(op(B), 'binary'))); b := [0 $ ('k' = 1 .. 8 - nops(a)), op(a)]
else error "1st argument must be symbol/string of length 1 or integer list,
but has received <%1>", B
end if;
if type(bits, 'list('integer')) and belong({op(bits)}, 1 .. 8) then [seq(b[k], k = {op(bits)})]
```

```

elif type(bits, 'list'('equation')) and belong({seq(lhs(k), k = bits)}, 1 .. 8) and
belong({seq(rhs(k), k = bits)}, 0 .. 1) then seq(assign('b'[lhs(bits[k])] = rhs(bits[k])),
    k = 1 .. nops(bits)); convert(parse(cat(op(map(convert, b, 'string')))), 'decimal', 'binary')
elif type(bits, 'procedure') then convert(parse(cat(op(map(convert, bits(b), 'string')))),
    'decimal', 'binary')
else error "2nd argument <%1> is invalid", bits
end if
end proc
Bit1 := proc(B::{integer, string, symbol}, bits::{procedure, list({integer, equation}})
local a, b, c, k;
    c := (x) -> [seq(parse(x[k]), k = 1 .. length(x))];
    if type(B, {'symbol', 'string'}) then a := c(cat("", convert(op(convert(B, 'bytes')), 'binary')));
        b := [0 $ ('k' = 1 .. 8 - nops(a)), op(a)]
    elif type(B, 'integer') then a := c(cat("", convert(B, 'binary'))); b := [0 $ ('k' = 1 .. 8 - nops(a)), op(a)]
    else error "1st argument must be symbol/string of length 1 or integer list,
        but has received <%1>", B
    end if;
    if type(bits, 'list'('integer')) then [seq(b[k], k = {op(bits)})]
    elif type(bits, 'list'('equation')) then seq(assign('b'[lhs(bits[k])] = rhs(bits[k])), k = 1 .. nops(bits));
        convert(parse(cat(op(map(convert, b, 'string')))), 'decimal', 'binary')
    elif type(bits, 'procedure') then convert(parse(cat(op(map(convert, bits(b), 'string')))),
        'decimal', 'binary')
    else error "2nd argument <%1> is invalid", bits
    end if
end proc
xbyte := proc(L::{list(binary)})
option remember;
    if nops(L) <> 8 then error "argument %1 is invalid", L else
        convert([convert(SN(cat("", op(L))), 'decimal', 'binary')], 'bytes')
    end if
end proc
xbyte1 := proc(L::{nonnegint, list(binary)})
local k;
option remember;
    if type(L, 'list') then xbyte([0 $ (k = 1 .. 8 - nops(L)), op(L)])
    elif xNB(L) and length(L) <= 8 then
        xbyte([0 $ (k = 1 .. 8 - length(L)), op(map(SN, convert(cat("", L), 'list1'))))
    else error "argument <%1> is invalid", L
    end if
end proc
xNB := (N::nonnegint) -> belong({op(map(SN, convert(cat("", N), 'list1')))}, {0, 1})

```

Typical examples of the procedure use:

```

> Bit('S', [k$k=1..8]), Bit([61], [1, 3, 8]), Bit([61], [2=1, 4=0, 8=0]), Bit("G", [6]),
Bit("0", [6=1, 7=1, 8=1]);

```

```

[0, 1, 0, 1, 0, 0, 1, 1], [0, 1, 1], 108, [1], 7

```

```
> R:= (L::list) -> [L[nops(L)-k]$k=0..nops(L)-1]: Bit("G", [k$k=1..8]), Bit("G", R),
  Bit("S", R), Bit([61], R);
```

[0, 1, 0, 0, 0, 1, 1, 1], 226, 202, 188

In particular, the given example illustrates use as the second actual *bits* argument of the **R**-procedure providing the inversion of lists:

```
> K:= (L) -> [L[8],L[2],L[6],L[4]+1 mod 2, L[5]+1 mod 2,L[3],L[7],L[1]]: map(Bit1, [61,56,36,
  40,7,14], K);
```

[164, 4, 60, 20, 186, 50]

```
> map(xbyte, [[1,1,0,1,0,1,0,1], [0,0,1,0,0,1,0,1]]);
```

["X", "%"]

```
> B:=Bit("G", [k$k=1..8]); "G"=xbyte(Bit(G,[k$k=1..8])), Bit(xbyte(B), [k$k=1..8])=B;
```

B := [0, 1, 0, 0, 0, 1, 1, 1]

"G" = "G" [0, 1, 0, 0, 0, 1, 1, 1] = [0, 1, 0, 0, 0, 1, 1, 1]

```
> map(xbyte1, [101101, 1110101, 11101011, 11011, 1011001, 1001010, 10001001, 10000001, 10101]);
```

["-", "u", "π", "\e", "Y", "J", "%o", "Í", "□"]

```
> map(xbyte1, [1010101,11010101,11100011,110110,10001001,10000001,1010101,11111001]);
```

["U", "X", "r", "6", "%o", "Í", "U", "π"]

```
> map(xNB, [10011,11001101,1102011,10101,1001010011101,11100011,110110,10001001]);
```

[true, true, false, true, true, true, true, true]

### boolop - basic functions/operators of Boolean algebra

Call format of the variables exported by the module:

**boolop**:- X(*args*) or with(**boolop**): X(*args*)

Formal arguments of the variables calls exported by the module:

**X** - name of a function/operator exported by the **boolop** module

*args* - formal arguments corresponding to the exported function/operator

Description of the module:

The module **boolop** exports a series of the useful functions/operators intended for providing of the basic operations of Boolean algebra, namely:

**&andB**, **&orB**, **&xorB**, **&notB**, **&impB**

where **&andB** - *n*-ary logical function/operator **and**, **&orB** - *n*-ary logical function/operator **or**, **&xorB** - *n*-ary logical function/operator **xor**, **&notB** - unary logical function/operator **not**, **&impB** - *n*-ary logical function/operator of implication (*implies*).

```
boolop := module ()
```

```
export `&andB`, # n-ary logical function/operator and
```

```
`&orB`, # n-ary logical function/operator or
```

```
`&xorB`, # n-ary logical function/operator xor
```

```
`&notB`, # unary logical function/operator not
```

```
`&impB`, # n-ary logical function/operator of implication (implies)
```

```
description "Basic logical operators/functions with Maple of releases 6, 7, 8, 9 and 10";
```

```
options `Copyright (c) RANS_IAN = Tallinn-Grodno-Vilnius; 14.06.2005`, package;
```

```
`&andB` := () -> `if` (nargs = 0, ERROR("arguments are missing"), `if` (belong({args}, 0 .. 1),
```

```
`if` (+` (args) = nargs, 1, `if` (member(1, {args}), 1, 0)), ERROR("arguments %1 are invalid",
```

```
{args} minus {0, 1}));
```

```
`&orB` := () -> `if` (nargs = 0, ERROR("arguments are missing"), `if` (belong({args}, 0 .. 1),
```

```

    if (^+(args) = nargs, 1, 0), ERROR("arguments %1 are invalid", {args} minus {0, 1}));
`&xorB` := () -> `if` (nargs = 0, ERROR("arguments are missing"), `if` (belong({args}, 0 .. 1),
    `if` (parse(cat(seq(`if` (args[k] = 0, " true xor", " false xor"), k=1 .. nargs))[1 .. -5]) = `true`, 0, 1),
    ERROR("arguments %1 are invalid", {args} minus {0, 1}));
`&notB` := (x::{0, 1}) -> x+1 mod 2;
`&impB` := (x::{0, 1}, y::{0, 1}) -> `if` ([x, y] = [0, 1], 1, 0);
end module

```

Типичные примеры применения средств, экспортируемых модулем:

```

> with(boolop); ⇒ [&andB, &impB, &notB, &orB, &xorB]
> 0 &andB 0, 1 &andB 1, 0 &andB 1, 1 &andB 0, `&andB`(1, 0, 1, 0, 1, 0); ⇒ 0, 1, 1, 1, 1
> 0 &orB 0, 1 &orB 1, 0 &orB 1, 1 &orB 0, `&orB`(1, 0, 1, 0, 1, 0); ⇒ 0, 1, 0, 0, 0
> 0 &xorB 0, 1 &xorB 1, 0 &xorB 1, 1 &xorB 0, `&xorB`(1, 0, 1, 0, 1, 0); ⇒ 1, 1, 0, 0, 0
> [`&notB`(0), `&notB`(1)], [0 &impB 0, 1 &impB 0, 0 &impB 1, 1 &impB 1]; ⇒ [1, 0], [0, 0, 1, 0]

```

*xpack* - a packing/unpacking of the positive integers

Call format of the procedure:

**xpack(N)**

Formal arguments of the procedure:

N - a positive integer or string

Description of the procedure:

The *xpack(N)* procedure, working according to the principle of *switch*, provides a packing of positive integers (*with providing of the subsequent restoration*) by means of putting of two decimal numerals in one byte. On a positive integer as the actual **N** argument, the procedure returns the string equivalent of the number **N**, while on the equivalent the number represented by it is being returned. However, the given operation is not reversible, i.e. not each string is an equivalent of some positive integer, as that illustrates the last example of use of the *xpack* procedure of a fragment represented below. Namely, the following relation takes place: *xpack(xpack(N)) = N*, where **N** - a positive integer; this relation, in general, is incorrect for arbitrary string **N**, excluding a case when **N** is a result of the procedure call *xpack(M)*, where **M** - a positive integer. The *xpack(N)* procedure essentially uses the *Bit* and *xbyte* procedures considered above. The examples represented below illustrate use of this procedure for a packing/unpacking of information.

```

xpack := proc(N::{nonnegint, string})
local a, b, c, d, k, s, p, h, q, Tab_con;
option remember;
    Tab_con := table([0 = [0, 1, 1, 0], 1 = [0, 1, 1, 1], 2 = [1, 0, 0, 0], 3 = [1, 0, 0, 1], 4 = [1, 0, 1, 0],
        5 = [1, 0, 1, 1], 6 = [1, 1, 0, 0], 7 = [1, 1, 0, 1], 8 = [1, 1, 1, 0], 9 = [1, 1, 1, 1]]);
    assign(q = "", h = [p $ (p = 1 .. 8)]);
if type(N, integer) then assign67(a = cat("", `if` (type(length(N), even), N, cat(`0`, N))), s = "");
    for k by 2 to length(a) do assign67('b' = SN(a[k .. k + 1])), assign67('c' = trunc(1/10*b));
        assign67('b' = b - 10*c), assign67('d' = [op(Tab_con[c]), op(Tab_con[b])]);
        assign67('s' = cat(s, xbyte(d)))
    end do;
    RETURN(s)
else
    for k to length(N) do assign67('a' = Bit(N[k], h));

```

```

    assign('b' = RTab(Tab_con, a[1 .. 4]), 'c' = RTab(Tab_con, a[5 .. 8]));
    assign67('q' = cat(q, b, c))
  end do;
  RETURN(SN(q))
end if
end proc

```

Typical examples of the procedure use:

```

> assign('G' = map(xpack, [0,1942,2004,14062004])), assign('S' = map(xpack, G)), G, S;
    ["f", "□", "tj", "zl†j"], [0, 1942, 2004, 14062004]
> xpack(424706762089961), xpack(xpack(424706762089961));
    "jЉЦИИняЗ", 424706762089961
> xpack(xpack("Grodno")), xpack(05240001400453510549041414), xpack(xpack("Tallinn"));
    "gvцно", "кЉбfg†j№ k†jzz", "llinn"
> xpack(%[2]), xpack("RANS_IAN"), xpack(8938), xpack(1302198904081996),
    xpack("yh□пjn□Ъ");
    524000141004535105490414140, 8938, "пћ", "yh□пjn□Ъ", 1302198904081996

```

### logbytes – basic logical operations with bytes

Call format of the procedure:

**logbytes(O::symbol {, args})**

Formal arguments of the procedure:

**O** – basic logical operation {*may be* {`xor`, `and`, `not`, `or`, `implies`}}

**args** – (*optional*) sequence of bytes of type {string, symbol}

Description of the procedure:

Procedure *logbytes* provides performance of basic logical operations {'xor', 'and', 'not', 'or', 'implies'} with bytes of type {symbol, string}. Logical operations are done with the corresponding bits of bytes defined by *args*-sequence of bytes. The result-byte is returned in *string*-format. Are supposed *n*-ary operations {'xor', 'and', 'or'}, binary operation 'implies' and unary operation 'not'. Procedure *logbytes* processes the basic especial and erroneous situations and uses two procedures *Bit* and *xbyte1*, and the module *boolop* which are considered above.

```
logbytes := proc(O::symbol)
```

```
local a, k, j;
```

```
if nargs = 1 then error "number of arguments must be > 1"
```

```
elif not member(O, {'xor', `and`, `not`, `or`, `implies`})
```

```
then error "1st argument must be {'xor', `and`, `not`, `or`, `implies`} but has received <%1>", O
```

```
else a := {};
```

```
for k from 2 to nargs do
```

```
if (proc(x) try type(x, {'symbol', 'string'}) and type(x, 'byte') catch : return false end try
```

```
end proc)(args[k]) then next
```

```
else a := {op(a), k}
```

```
end if
```

```
end do
```

```
end if;
```

```
if a <> {} then error "arguments with numbers %1 must be bytes of type {string, symbol}", a
```

```
else a := map(Bit, [args[2 .. -1]], ['k' $ ('k' = 1 .. 8)])
```

```

end if;
if O = `and` then xbyte1([seq(boolop[`&andB`](seq(a[k][j], k = 1 .. nops(a))), j = 1 .. 8)])
elif O = `or` then xbyte1([seq(boolop[`&orB`](seq(a[k][j], k = 1 .. nops(a))), j = 1 .. 8)])
elif O = `xor` then xbyte1([seq(boolop[`&xorB`](seq(a[k][j], k = 1 .. nops(a))), j = 1 .. 8)])
elif O = `not` then xbyte1([seq(boolop[`&notB`](a[1][j]), j = 1 .. 8)])
elif O = `implies` and 2 < nargs then xbyte1([seq(boolop[`&impB`](seq(a[k][j], k=1..2)), j = 1..8)])
else 'procname(args)'
end if
end proc

```

Typical examples of the procedure use:

> **logbytes**(non, A, V, Z, A, G, N, V, S, V);

Error, (in *logbytes*) 1st argument must be {`and`, `or`, `xor`, `not`, `implies`} but has received <non>

> **logbytes**(implies, A); ⇒ logbytes(implies, A)

> **map**(logbytes, [and, or, xor, not, implies], W, V, R, Q, N, K, T, R, U, S, A);  
 ["\_", "@", "F", "", ""]

> **logbytes**(implies, [1, 0, 0, 0, 1, 0, 0, 1], v, z, A, g, n);

Error, (in *logbytes*) arguments with numbers {2} must be bytes of type {string, symbol}

Taking into account the important role of the *bit-by-bit* information processing in the *Maple* environment, it is supposed, the tools represented in the present section, should be essentially extended with increase of their reactivity and functionality.

It is necessary to be surprised only how in mathematically oriented package such means were found oneself outside of a field of vision of developers. Up to that, the reader for a programming of algorithms dealing with bit-by-bit information processing, can use means of the present chapter of the book or use them as a basis for creation of own similar means. Experience of use of the suggested means has shown their quite satisfactory efficiency at the decision of a lot of problems bit-by-bit information processing and in problems of mathematical logic.

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

# Chapter 5.

## Software that extends and improves the standard tools of *Maple*

In the given chapter, the tools that extend and improve the standard *Maple* tools for releases 6 – 10 are represented. These tools are used enough widely both at operation with the *Maple* package in interactive mode and at programming of various problems in its environment. They represent undoubted interest at programming of various problems in the *Maple* environment, both by simplifying the programming and by making it by more clear. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

### **WARNING** – *expansion of the standard procedure WARNING*

Call format of the procedure:

**WARNING**(*args*)

Formal arguments of the procedure:

*args* – sequence of formal arguments analogous to the standard procedure **WARNING**

Description of the procedure:

A call to the **WARNING** procedure causes a warning to be produced on the current output stream. The message is displayed preceded by the string "Warning, ...". However, program processing of such message is rather difficult. With the purpose of a possibility of processing of messages, output by the standard **WARNING**-procedure we have created a simple enough procedure of the same name **WARNING** having the identical set of formal arguments with standard procedure and returning the messages identically to it. However, as against the first procedure, our procedure **WARNING** through the global *\_warning*-variable returns the text of the output messages, providing their program processing. Value of the *\_warning*-variable has *string*-type. The examples presented below well illustrate the told.

```
WARNING := proc(msg::{string, symbol})
local a, b, c, d, k;
global _warning;
  print(INTERFACE_WARN(1, args)), assign(a = [parse(seqstr(args))],
    c = (x -> `if` (search(seqstr(x), ":-", 'd'), convert(seqstr(x)[d + 2 .. -1], 'symbol'), x)));
  if nops(a) = 1 then assign('_warning' = cat("", a[1]))
  else b := nops(a) - 1;
    assign('_warning' = cat("", sub_1(["%0" = seqstr(op(map(c, a[2 .. -1]))),
      seq(cat("%", k) = convert(c(a[k+1]), 'string'), k = 1 .. b)], a[1])))
  end if
end proc
```

Typical examples of the procedure use:

```
> WARNING("AVZ - <%1>, AGN - <%2>, VSV - <%3>, Art - <%4>, Kr - <%5>", 63,58,38,16,9);
Warning, AVZ - <63>, AGN - <58>, VSV - <38>, Art - <16>, Kr - <9>
> _warning; ⇒ "AVZ - <2>, AGN - <3>, VSV - <4>, Art - <5>, Kr - <6>"
> WARNING("AVZ - <%1>, AGN - <%2>, VSV - <%3>, Art - <%4>, Kr - <%5>", sin(x), [a, b],
sqrt(x+y), 16, c/d);
Warning, AVZ - <sin(x)>, AGN - <[a, b]>, VSV - <(x+y)^(1/2)>, Art - <16>, Kr - <c/d>
> _warning;
"AVZ - <sin(x)>, AGN - <[a, b]>, VSV - <(x+y)^(1/2)>, Art - <16>, Kr - <c/d>"
> with(linalg): _warning;
Warning, the protected names norm and trace have been redefined and unprotected
"the protected names norm and trace have been redefined and unprotected"
> with(SimpleStat);
Warning, these previously assigned names now have a global binding: ACC, FD,
LRM_NRM, LT, MAM, Weights
[ACC, CC, CR, Ds, FD, LRM_NRM, LT, MAM, MCC, PCC, SR, Sko, Weights]
> _warning;
"these previously assigned names now have a global binding: ACC, FD, LRM_NRM, LT, MAM,
Weights"
```

### **tpr** - typification of result of procedure call

Call format of the procedure:

**tpr**(P::*procedure*, V::*anything*)

Formal arguments of the procedure:

**P** - a procedure

**V** - an arbitrary *Maple* expression

Description of the procedure:

The package has the advanced enough means of typification of *Maple*-objects. Here we present one non-standard approach of typification of the result returned by procedure. The given approach is based on use of the (::)-operator of typification directly behind the opening **proc**-bracket of procedure as it illustrate examples presented below.

Use of the given operator enables to receive by means of *op* (-1, eval (A)) a construction coded behind the (::)-operator, ascribed to the opening procedural bracket. The given reception has many interesting outlets, however here we shall present only simple procedure **tpr**(P, res) that converts res-result of call of any P-procedure to the type determined by the above way. The fragment presented below transparently enough illustrates the suggested approach. As any correct *Maple*-expression can be used with the (::)-operator, and also their sequence, the given reception can have many interesting appendices. The interested user can find out mass of interesting and useful appendices of the opportunity suggested above of "typification" of **proc**-brackets of procedure which, generally, depends on release of the package as stated above.

```
tpr := proc(pn::procedure, R::anything) convert(R, op(-1, eval(pn))) end proc
```

Typical examples of the procedure use:

```
> A:= proc(x,y,z)::float; option `Copyright RANS`; local a; tpr(procname, `(args)*a) end proc;
A := proc(x, y, z)::float; ... end proc
> A(9/16, 58/63, 39/42); ⇒ 2.411706349*a
```

```
> B:= proc(x, y, z)::float, (x+y+z)/(a+b+c)^2, agn; local a; op(-1, eval(procname))[2] end proc:  
> B(63.42, 58.47, 38.67); => (x + y + z)/(a+b+c)^2  
> P:=proc():string; tpr('procname', '+'(args)) end proc: P(63,42,58,47,38,67,16,89,9,96); => "525"
```

**Save** - extended saving of the Maple procedures and program modules

**Save1**

**Save2**

**savem**

**savemp**

**savema**

**savemu**

Call format of the procedures:

**Save(P, F)**

**Save1(P, F)**

**Save2(P, F)**

**savem(M, G)**

**savemp(M1, G)**

**savema(args)**

**savemu(M1, G)**

Formal arguments of the procedures:

**P** - a list of names of the Maple procedures or/and program modules assigned to saving

**F** - a list of files assigned to receiving of the above Maple objects

**M** - a program module assigned to saving

**G** - a *m*-file assigned to receiving of program modules

**M1** - a program module, set or list of program modules assigned to saving

**args** - names sequence in which the last element defines a file

Description of the procedures:

The *Save* procedure (in contrast to the built-in function `save`) is used to save the Maple objects (*procedures* or/and *program modules*) defined by the first actual **P** argument in files defined by the second actual **F** argument. Between elements of the lists **P** and **F** should be one-to-one correspondence, namely: the *j*th object of **P** is saved in the *j*th file of **F**. In a case of violation of such coincidence the error is arises. In addition, if path to a receiving file does not exist, then it is created with output of appropriate warning, if it is necessary; this note is valid for all tools of so-called **Save**-group. For these purposes, the mechanism of the *pathtf* procedure is used.

The built-in function `save` of Maple saves program modules incorrectly. An quite detailed discussion of this question can be found in our books [10,29-33,43,44]. By reason of that the *Save* procedure saves both the procedures and the program modules defined by actual **P** parameter in files of the Maple language format, whereas the procedures in files of both formats. If the procedure call *Save* defines a saving of a program module in file of internal Maple format (*m*-file), then the module will be saved in file of the input Maple language format with main name of the target file and with *txt*-extension. In this case, an appropriate warning is output.

Extremely useful procedure *Save1* essentially extends both the built-in function `save`, and the above *Save* procedure. The extension of the procedure allows to save the groups of program objects (*procedures* and *program modules*) in separate files. In this case the first actual **P** argument of the procedure call *Save1(P, F)* can be represented by a list, whose elements are the lists or the sets of names of the saved objects (whose definitions were evaluated in the current session or are in a library logically linked with the main Maple library), whereas the second argument **F** is analogous to the second argument of the *Save* procedure represented above. The *Save1* procedure in a series

of cases essentially simplifies programming problems dealing with organization of storage of procedures and program modules in the peripheral memory. The detailing of description of the *Save1* procedure can be found below in an example illustrating its application.

At last, the *Save2* procedure quite essentially extends the built-in function ``save`` with complete inheriting of syntax of its call. The *Save2* procedure uses the same formal arguments, as the built-in function ``save``, but in contrast to the second it allows correctly to save in the *m*-files the variables, procedures, and program modules. The procedure call returns full path to a target datafile.

For saving of variables, they at a procedure call should be coded in *single quotes* (for example, 'X'). The procedure can output the warnings of a view *"Warning, Incorrect saving of modules is very likely!"*, notifying that during a saving the procedure has met program modules, which can be incorrect, for example, variables, exported by them, are not defined. The given warning has rather notifying character, and at a priori correctness of modules it does not influence the correctness of their saving in files of both formats (*the input Maple language format and the internal Maple format*). The detailing of description of the *Save2* procedure can be found below in an example illustrating its application.

Procedure *savem(M, G)* implemented by one-line expra-code, provides saving of the program module defined by first actual argument **M**, in file **G** of the internal *Maple* format, i.e. in a *m*-file with return of *NULL* value, i.e. nothing. as a result of the subsequent reading of the given *m*-file by means of clause `read` the **M**-module saved in the file becomes completely accessible in the current session after call **M()** or **with(M)** as it illustrates last example of the fragment presented below.

Procedure *savemp(M1, G)* implemented by one-line expra-code, provides saving of the program modules defined by first actual argument **M1**, in file **G** of the internal *Maple* format, i.e. in a *m*-file with return of *NULL* value, i.e. nothing. As a result of the subsequent reading of the given *m*-file by means of clause `read` the **M1**-modules saved in the file becomes completely accessible in the current session after call **M1(): ... M1k():** as it illustrates last example of the fragment presented below.

Procedure *savema(args)* provides saving of names of the *Maple*-objects determined by the first *nargs-1* actual arguments, in a file of internal or input *Maple* format determined by last actual argument, with return of full path to the receiving file. After the subsequent reading of the given file by the statement `read` the module (*modules*) saved in the file become completely accessible in the current session. For providing of access to the **M** module saved thus, before its first use, the call **M()** is required. Further manipulations with the module are executed according to agreements of the package.

At last, procedure *savemu(M1, G)* implemented by one-line expra-code, provides saving of the program modules defined by first actual argument **M1**, in file **G** of the internal *Maple* format, i.e. in a *m*-file with return of *NULL* value, i.e. nothing. As a result of the subsequent reading of the given *m*-file by means of statement `read` the **M1**-modules saved in the file becomes accessible relative to their exports. Access to exports `exp` of the **M1**-module saved thus is provided with call **M1[exp](args)**. Alongside with statement `read`, reading of the *m*-file created by procedure *savemu*, is supported also by procedures *read3* and *readmp*. Meanwhile, it is necessary to note, that such objects are not recognized by means of the package as the program modules, as illustrate last examples of the fragment presented below.

Thus, the represented group of procedures *Save*, *Save1*, *Save2*, *savem*, *savemp*, *savema*, *savemu* and *Read1* fulfills a much wider set of functions, than 2 statements `save` and `read`. The examples of joint appendices of the above procedures and `read` represented below illustrate the told above.

```
Save := proc(P::list({symbol}), F::list({string, symbol}))
local a, b, c, d, k, h, f;
  if P = [] or F = [] then error "one or both actual arguments are empty" else
```

```

    assign(a = nops(P), d = nops(F)) end if;
if a <> d then error "disparity of quantities of procedures and datafiles: %1 <> %2", a, d end if;
for k to a do assign('c' = pathtf(F[k])); if not type(eval(P[k]), {'procedure', `module`}) then
    WARNING("<%1> is not a procedure and not a module", P[k]); next
else
    if c[-2 .. -1] = ".m" and type(P[k], `module`) then fremove(c); c := cat(c[1 .. -3], ".txt");
    WARNING("module <%1> has been saved in datafile <%2>", P[k], c) end if;
    assign('f' = cat(currentdir(), "/"$Art16_Kr9$));
    null(writebytes(f, cat("save(", P[k], ", ", "", CF1(c), "", ":)"), fclose(f)),
    (proc() read f; fremove(f) end proc())
end if
end do
end proc
Save1 := proc(P::list({symbol, list(symbol), set(symbol)}), F::list({string, symbol}))
local a, d, k, j, h, f, Q, `_$avz_agn_sv_art_kr_arn$`, R;
if P = [] or F = [] then error "one or both actual arguments are empty" else
    assign(a = nops(P), d = nops(F), f = map(pathtf, F))
end if;
if a <> d then error "disparity of quantities of procedures and datafiles: %1<>%2", a, d end if;
if nops(f) <> nops(map(CF, {op(f)}, 'string')) then WARNING("multiple names in %1", f) end if;
for k to nops(P) do
    assign('R' = [seq(`if` (member(whattype(eval(P[k][j])), {'procedure', `module`}), P[k][j],
    null(WARNING("<%1> is unassigned or a variable", P[k][j])), j = 1 .. nops(P[k])),
    `_$avz_agn_sv_art_kr_arn$` = 1942194767899662);
    save `_$avz_agn_sv_art_kr_arn$`, f[k];
    `if` (cat(" ", f[k])[-2 .. -1] = ".m", seq(`if` (type(eval(R[j]), `module`), [mod21(R[j]), modproc(R[j])],
    NULL), j = 1 .. nops(R)), NULL);
    `if` (cat(" ", f[k])[-2 .. -1] <> ".m", seq(`if` (type(eval(R[j]), `module`), mod21(R[j]), NULL),
    j = 1 .. nops(R)), NULL);
    assign('Q' = [seq(cat(`$Art16_Kr9$`, j), j = 1 .. nops(R))], seq(Save([R[j]], [Q[j]]), j = 1 .. nops(R));
    seq(MmF(f[k], Q[j]), j = 1 .. nops(R)), fremove(op(Q))
end do
end proc
Save2 := proc()
local a, b, c, j, k, G, S, nu, f, omega, h;
if nargs < 2 then return NULL else nu := interface(warnlevel);
    omega := (x) -> interface(warnlevel = x) end if;
if type(args[-1], 'file') then f := args[-1]; omega(0) elif not type(eval(args[-1]), {'string', 'symbol'})
then error "<%1> cannot specify a datafile", args[-1] else omega(0); f := Mkdir(args[-1], 1)
end if;
    assign(h = cat(currentdir(), "/"$Art16_Kr9$));
    assign(G = [], S = ""), omega(2), assign(a = {op(SLD(convert('procname(args)', 'string')[7 .. -1], ",")
    [1 .. -2]))}; seq(`if` (type(eval(args[j]), {'procedure', `module`}), assign('G' = [op(G),
    args[j]]), NULL), j = 1 .. nargs - 1), assign(b = a minus {op(map(convert, G, 'string'))});
if b = {} and G = [] then return omega(nu) end if;

```

```

if b <> {} then assign(c = cat("_Vars_", " := proc() global ", seq(cat(b[j], ", "), j = 1 .. nops(b)),
"; assign(", seq(cat(convert(cat(" ", b[j], " ") = eval(convert(b[j], 'symbol')), 'symbol'), ` `),
j = 1 .. nops(b)), " ) end proc:")), seq(assign('S' = cat(S, `if` (member(k, {op(Search2(c, {"\ ",
";", ", ,")"}))), NULL, c[k])), k = 1 .. length(c)), writebytes(h, S), fclose(h),
  (proc() read h; remove(h) end proc)()
end if;
Save1([op(G), `if` (b = {}, NULL, _Vars_)], [f], omega(nu), f
end proc
savem := (M::`module`, F) -> op([mod21(M), assign(`000` = cat("assign(", M, "=", convert(eval(M),
'string'), "))), assign(M = op([proc() parse(args) end proc(`000`), assign(`111` = convert(M,
'string')), 'M'])), (proc() save(M, `111`, F) end proc)(), unassign(`000`, `111`));
savemp := (M::{`module`, set(`module`), list(`module`)}, F) -> (proc() local a;
  a := (x) -> `if` (type(x, `module`), [x], x); map(mod21, a(M)); modproc(op(a(M)));
  (proc() save args, F end proc)(op(a(M)))
end proc()
savema := proc()
local a, b, c, k;
  if nargs < 2 then error "procedure call must contain at least two arguments"
  elif not type(args[-1], {'symbol', 'string'}) then
    error "last argument must be string or symbol"
  else
    if not type(args[-1], 'file') then k := interface(warnlevel); interface(warnlevel = 0);
      c := Mkdir(args[-1], 1); interface(warnlevel = k)
    else c := args[-1]
    end if;
    assign(a = {}, b = {})
  end if;
  seq(`if` (type(k, `module`), assign('a' = {op(a), k}),
    `if` (type(k, 'symbol'), assign('b' = {op(b), k}), NULL)), k = [args[1 .. -2]]);
  if a <> {} then map(mod21, a); modproc(op(a)) end if;
  if a union b = {} then error "procedure call has not objects for saving"
  else (proc() save args, c end proc)(op(a union b)), c
  end if
end proc
savemu := (M::{`module`, set(`module`), list(`module`)}, F) ->
(proc()
local a, b;
  assign(a = ((x) -> `if` (type(x, `module`), [x], x)));
  map(mod21, a(M)), seq(parse(cat(" ", b, " := eval('parse'(convert(eval(" , b, " ), 'string')):")),
'statement'), b = a(M));
  (proc() save args, F end proc)(op(a(M)))
end proc()

```

Typical examples of the procedure use:

```

> P:=() -> `+(args)/nargs: M:=module() export x; x:=() -> `+(args)/nargs end module: F1:=
`C:/AVZ/rans`: F2:=`C:/AVZ/IAN.m`: F3:=`C:/AVZ/IAN`: Save([P,M,M], [F1,F2,F3]);
Warning, module <M> has been saved in datafile <c:\avz\ian.txt>

```

Application of the procedure *Save1* for saving of the procedures **PP**, **PP1**, **PP2** and program modules **SVEGAL**, **GS**, **MM**, **MM1** of two types in two files of various formats (the input *Maple* format and the internal *Maple* format) with the subsequent check of a saving result by means of the procedure *Read1*. Of this example, the saving principle realized by the *Save1* procedure is well being viewed, namely:

- (1) the procedures save own type regardless of a type of a receiving file;
- (2) in case of a receiving file of the input *Maple* format, the program modules of the second type are converted into modules of the first type (*what will provide their subsequent correct use*);
- (3) in case of receiving file of the input *Maple* format, the program modules of both types are converted into procedures (*what will provide their subsequent correct use by a special syntax of the first access to variables exported by them*).

The first access to variables exported by a module, saved thus, has the following simple view:

**Name(): Name:- S(args)**

where: *Name* - a name of a saved program module, *S* - a variable exported by it, and *args* - actual arguments passed to the variable. The following fragment illustrates some appendices of the *Save1* procedure.

```
> MM:= module() export x,m; m:= module() export y; y:=() -> `+(args) end module end
module: MM1:= module() local F,m; export a; option package; m:= module() export y;
y:=() -> `+(args) end module; F:=() -> m:- y(args); a:=F end module: module SVEGAL()
export x; x:=() -> `+(args)/nargs^6 end module: PP:=proc() `+(args)/nargs^2 end proc:
PP1:=proc() `+(args)/nargs end proc: PP2:=proc() `+(args)/nargs end proc: module GS()
export x; x:=() -> `+(args)/nargs^3 end module:
> Save1([[MM,PP,PP2,GS], {MM1,SVEGAL,PP1}], ["C:/temp/VSG", "C:/temp/VSG.m"]);
Warning, Incorrect saving of modules is very likely!
> restart; Read1(["C:/temp/VSG", 'h'), eval(h);
table([procs = {PP, PP2}, mods2 = {}, mods1 = {GS, MM}, vars = {}])
> restart; Read1(["C:/temp/VSG.m", 'h'), eval(h); A:= 2*6^5;
table([procs = {SVEGAL, PP1, MM1}, mods2 = {}, mods1 = {}, vars = {}])
> SVEGAL(): A*SVEGAL:- x(61,56,36,40,6,14), 3*A*SVEGAL:- x(42,47,67,62,96,89);
71, 403
> MM1(): MM1:- a(61,56,36,40,6,14), MM1:- a(42,47,67,62,96,89);
213, 403
```

Application of the procedure *Save2* for a saving of the procedures **PP**, **PP1**, the program modules **GS**, **MM**, **SVEGAL**, **MM1** of two types, and variables {**x**, **y**, **x**, **t**, **v**, **u**, **m**} in a file of the internal *Maple* format (*m*-file) with the subsequent check of a saving result by means of the built-in function *read*. Of this example, the saving principle realized by the *Save2* procedure, is enough well being viewed, namely:

- (1) the procedures save own type regardless of a type of a receiving file;
- (2) the variables are saved in a receiving file as procedures;
- (3) in case of receiving file of the input *Maple* format, the program modules of the second type are converted into modules of the first type (*what will provide their subsequent correct use*);
- (4) in case of receiving file of the internal *Maple* format, the program modules of both types are converted into procedures (*what will provide their subsequent correct use by a special syntax of the first access to variables exported by them*).

The first access to variables exported by a program module, saved thus, and next read by the built-in *read* function has the following simple view:

**Name(): Name:- S(args)**

where: *Name* - a name of a saved program module, *S* - a variable exported by it, and *args* - actual arguments passed to the variable. While the variables, saved by the *Save2* procedure, after reading of *m*-file containing them, become accessible in the current session after the unitized call `_Vars_() { : | ; | , }`. While a reading of the objects saved by the *Save2* procedure, by means of the *Read1* procedure already at the first access to them suppose the standard *Maple* syntax. The following fragment illustrates some appendices of the *Save2* procedure.

```
> MM:= module() export x,m; m:= module() export y;y:= () -> `+(args) end module end
module: MM1:= module() local F,m; export a; option package; m:= module() export y;
y:=() -> `+(args) end module; F:=() -> m:- y(args); a:=F end module: module SVEGAL()
export x; x:=() -> `+(args)/nargs^6 end module: PP:= proc() `+(args)/nargs^2 nd proc:
PP1:=proc() `+(args)/nargs end proc: PP2:=proc() `+(args)/nargs end proc: module GS()
export x; x:=() -> `+(args)/nargs^3 end module: x:=61: y:=56: z:=36: t:=67: v:=89: u:=96:
m:=6: Save2('x',MM,'y',MM1,'z',SVEGAL,'t',GS,'v',PP,'u',PP1,'m', "C:/Temp/AVZ.m");
```

Warning, Incorrect saving of modules is very likely!

```
> restart; Read1(["C:/Temp/AVZ.m"], 'h'), eval(h);
table([mods1 = {}, procs = {PP1, MM, MM1, GS, SVEGAL, _Vars_, PP}, vars = {}, mods2 = {}])
> [x, y, z, t, v, u, m], 36*PP(42,47,67,62,89,96), SVEGAL:-`x`(33);
[61, 56, 36, 67, 89, 96, 6], 403, 33
> restart; read("C:/Temp/AVZ.m"); SVEGAL(): [SVEGAL:-`x`(33)], _Vars_(), x,y,z,t,v,u,m;
[33], 61, 56, 36, 67, 89, 96, 6
```

**renmf** - replacement of names of the objects saved in *m*-files of the internal *Maple* format

Call format of the procedure:

**renmf**(F::file, N::symbol, M::symbol)

Formal arguments of the procedure:

**F** - name of a file of the internal *Maple* format (*m*-file) or full path to it

**N** - name of an object saved in file **F** which must be replaced

**M** - a new name for the above object **N**

Description of the procedure:

In a series of cases at work with *m*-files there is a necessity to rename objects contained in such file without loading such files into the current *Maple* session with the subsequent saving of the renamed objects. In this case the procedure *renmf* can be useful enough.

The procedure call *renmf*(**F**, **N**, **M**) provides renaming of all objects of *m*-file **F** with name **N** by a new name **M**. In addition, loading of file **F** into the current *Maple* session is not made. Successful procedure call returns the equation of the form **N** = **M**. In case of absence in file **F** of objects with required name **N**, the procedure call returns the *NULL* value, i.e. nothing, with output of the corresponding message which can be processed through the predetermined variable *\_warning* as illustrates the fragment presented below. Procedure handles the basic erroneous and especial situations. The examples presented below evidently enough illustrate the said.

```
renmf := proc(F::file, N::symbol, M::symbol)
local a, b;
if Ftype(F) <> ".m" then
error "the 1st argument must be m-file but has received %1-file", Ftype(F)[2 .. -1]
else
assign(a = readbytes(F, 'TEXT', 'infinity'), b = cat("\nI", Iddn(N), N)), close(F);
if search(a, b) then
```

```

    null(writebytes(F, [SUB_S([b = cat("\nI", Iddn(M), M)], a, 'sensitive'), close(F)][1]),
    close(F)), N = M
  else WARNING("file <%1> does not contain object <%2>", F, N)
  end if
end if
end proc

```

Typical examples of the procedure use:

```

> renmf("C:\\temp\\Paskula.m", RANS_IAN_RAC_REA, `Tallinn-Vilnius-Grodno`);
      RANS_IAN_RAC_REA = Tallinn-Vilnius-Grodno
> renmf("C:\\temp\\Paskula.m", `tallinn-Vilnius-Grodno`, RANS_IAN_RAC_REA);
  Warning, file <C:/temp/Paskula.m> does not contain tool <tallinn-Vilnius-Grodno>
> _warning;  => "file <C:/temp/Paskula.m> does not contain tool <tallinn-Vilnius-Grodno>"
> renmf("D:\\temp\\Maple.doc", `Tallinn-Vilnius-Grodno`, RANS_IAN_RAC_REA);
  Error, (in renmf) the 1st argument must be m-file but has received doc-file
> renmf("C:\\temp\\Paskula.m", `Tallinn-Vilnius-Grodno`, RANS_IAN_RAC_REA);
      Tallinn-Vilnius-Grodno = RANS_IAN_RAC_REA

```

### LatexI - replacement of imaginary unit in latex-documents

Call format of the procedure:

**LatexI**(s::equation({symbol, string}))

Formal arguments of the procedure:

s - equation  $a = b$  defining replacement of imaginary unit

Description of the procedure:

The *latex(expr)* function produces output which is suitable for printing the input *expr* with a *LaTeX 2e* processor. The function *latex* produces output as a side-effect, and returns *NULL* as the function value, i.e. nothing. By default, in complex expressions the *latex* function defines imaginary unit as the symbol "i". However, in a series of cases it is expedient to use other designation for imaginary unit. This problem can be solved by means of the procedure *LatexI*.

The procedure call *LatexI(a = b)* returns the *NULL* value, i.e. nothing, providing replacement of the current designation of imaginary unit "a" by a new designation "b". The given replacement has an effect until a new redefinition of imaginary unit by means of the corresponding procedure call, until performance of the command **restart**, or until reload of *Maple*. Dynamic updating of one of the *Maple* procedures on the basis of a method of disk transits is put in the base of the procedure algorithm. The procedure *LatexI* operates with *Maple* of release 6 and later.

```

LatexI := proc(s::equation({string, symbol}))
local a, b, c, d, f, k, t, p;
  assign(c = cat("", lhs(s))[1], p = cat("", rhs(s))[1], f = cat(currentdir(), "/_Art16_Kr9$_"));
  if Release1() = 6 then d:= {`*`}; t:= `latex/print`
  else d:= {`*`, `;`, `)`}; t:= `latex/latex/complex`
  end if;
  assign(a = convert(eval(t), 'string'), b = table([seq(cat(c, k) = cat(p, k), k = d)]));
  for k to length(a) - 1 do
    if member(a[k .. k + 1], map(op, {indices(b)})) then
      a := cat(a[1 .. k - 1], b[a[k .. k + 1]], a[k + 2 .. -1])
    end if
  end for
end proc

```

```

end do;
writeline(f, cat("", t, " := ", a, ":"), close(f);
(proc(x) read x end proc)(f), fremove(f)
end proc

```

Typical examples of the procedure use:

```

> latex(a+b*I), latex((a+b*I)/(sqrt(c-x*I*Pi))), latex(sin(x+Pi*I)+exp(a+5*I)), latex(gamma*I+c);
a+ib
{\frac {a+ib}{\sqrt {c-ix\pi }}}
\sin \left( x+i\pi \right) +{e^{a+5\,i}}
i\gamma+c
> LatexI(i = j); latex(a+b*I), latex((a+b*I)/(sqrt(c-x*I*Pi))), latex(sin(x+Pi*I)+exp(a+5*I)),
latex(gamma*I+c);
a+jb
{\frac {a+jb}{\sqrt {c-jx\pi }}}
\sin \left( x+j\pi \right) +{e^{a+5\,j}}
j\gamma+c

```

**savead** – *saving of the Maple objects in APPEND mode*

Call format of the procedure:

**savead(P, F)**

Formal arguments of the procedure:

**P** – a sequence of names of the *Maple* objects assigned to saving in *APPEND* mode

**F** – a file assigned to receiving of the above *Maple* objects

Description of the procedure:

The *savead* procedure (*in contrast to the built-in function `save` and our procedures of Save group*) is used to save the *Maple* objects defined by the first actual **P** argument in file defined by the second actual **F** argument in *APPEND* mode. For saving of the *Maple* objects their names at the procedure call should be encoded in single quotes (*for example, 'G'*). In addition, if path to a receiving file **F** does not exist, then it is created with output of the appropriate warning, if it is necessary. For these purposes the mechanism of the *pathif* procedure is used. Actual arguments of the procedure are coded similarly to the standard function *save*, on condition that names of the saved *Maple* objects should be encoded in single quotes. Useful procedure *savead* well supplements the built-in function *save* and the procedures *Save* [239, 240]. The procedure in a series of cases essentially simplifies programming problems dealing with organization of storage of the *Maple* objects in the peripheral memory.

```

savead := proc()
local a, b, c, d, h, f;
assign(b = cat(currentdir(), "/_Art16_Kr9$_",
`if (cat(" ", args[-1])[-2 .. -1] = ".m", ".m", ""), c = args[1 .. -2]);
(proc(c) save args, b end proc)(c), assign(f = pathif(args[-1]));
if cat(" ", f)[-2 .. -1] = ".m" then
if IsEmpty(f) then writebytes(f, readbytes(b, `infinity`)); return fremove(b), close(f)
end if;
assign(d = interface(warnlevel), h = cat(b[1 .. -3], "$.m")), interface(warnlevel = 0);
mmf(f, b, h), writebytes(f, readbytes(h, `infinity`));
fremove(h, b), close(f), interface(warnlevel = d)

```

```

else assign(a = fopen(f, 'APPEND')), writebytes(a, readbytes(b, `infinity`));
  fremove(b), close(a)
end if
end proc

```

Typical examples of the procedure use:

```

> p, n:=7, 100: t:=time(): for k to 1000 do P:=randpoly(x, degree=15, coeffs=rand(-n .. n), dense):
  if Primitive(P) mod p then savead('P', "C:\\temp\\ppolynom") end if end do: time() - t;
  2.220
> p, n:=7, 100: t:=time(): for k to 1000 do P:=randpoly(x, degree=15, coeffs=rand(-n .. n), dense):
  if Primitive(P) mod p then savead('P', "C:\\temp\\ppolynom.m") end if end do: time() - t;
  1.814
> savead('P', 'V', "C:\\temp\\file"), savead('P', 'V', "C:\\temp\\file.m");
> restart; read("C:\\temp\\ppolynom.m"); read("C:\\temp\\file.m");
> savead('P', 'V', "C:\\temp\\file1.m"); read("C:\\temp\\ppolynom");

```

``type/plotopt`` - check of a Maple object to be plot-option or plot3d-option

``type/plot3dopt``

Call format of the procedures:

`type(S, plotopt)`

`type(S, plot3dopt)`

Formal arguments of the procedures:

S - any allowable Maple-expression of equation-type

Description of the procedures:

For a lot of problems dealing with graphic objects the types *plotoption* and *plot3doption* seem useful enough which are absent in the package *Maple* of all current releases. The options to the plot procedure are given after the function(s), horizontal range, and vertical range, as equations of the form *option = value*. Options to the *plot3d* procedure are given after the first three arguments, as equations of the form *option = value*. The procedure call `type(S, plotopt)` returns the *true* value if S is a *plot*-option, and the *false* value otherwise. The procedure call `type(S, plot3dopt)` returns the *true* value if S is a *plot3d*-option, and the *false* value otherwise.

```

`type/plotopt` := proc(po::equation)

```

```

local a; try a := plot(L, po) catch: return false end try; true end proc

```

```

`type/plot3dopt` := proc(po::equation)

```

```

local a; try a := plot3d(1, 0 .. 1, 0 .. 1, po) catch: return false end try; true end proc

```

Typical examples of the procedures use:

```

> map(type, [thickness=2, color=blue, font=[TIMES, BOLD, 16]], 'plotopt'); => [true, true, true]
> map(type, [thicknes=2, color=aaa, fonts=[TIMES, BOLD, 16]], 'plotopt'); => [false, false, false]
> map(type, [filled=true, scaling=CONSTRAINED, style=WIREFRAME], 'plot3dopt');
  [true, true, true]
> map(type, [filled = FAIL, scaling = WIREFRAME, style = CONSTRAINED], 'plot3dopt');
  [false, false, false]

```

`plotpw` - an useful output of the piecewise functions

Call format of the procedure:

`plotpw(x {, [f(x), a, b {, plotopt}], ...} {, plotopt})`

Formal arguments of the procedure:

**x** - name of an independent variable  
**[f1(x), a1, b1 {, plotopt}]** - (optional) descriptor of function **F1(x)** on **[a1, b1]**-segment  
**plotopt** - (optional) graphic **plot**-options

Description of the procedure:

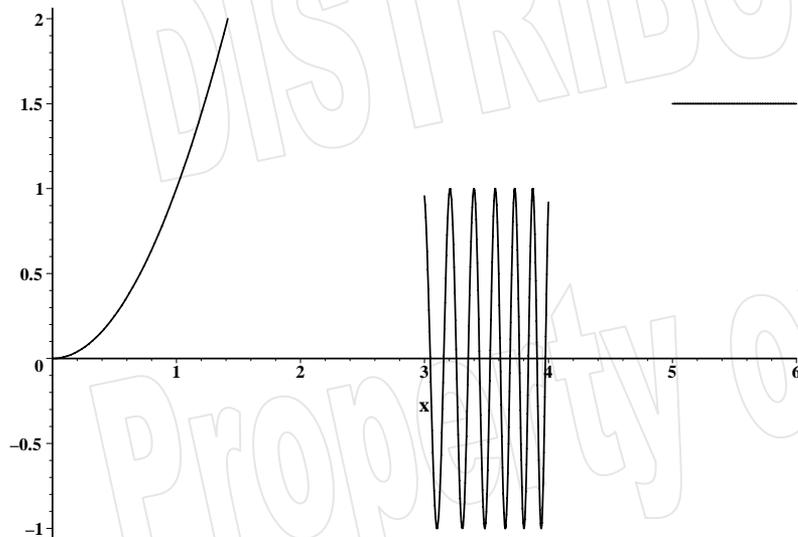
The procedure call **plotpw(x {, [f1 (x), a1, b1 {, plotopt}], ...} {, plotopt})** returns the graph of the piecewise function, whose functions **fj(x)** are defined on segments **[aj, bj]** of **x**-axis and which are designed according to the set of **plotopt**-options. **plotopt**-options given after group of descriptors of functions define designing for the graph of function as a whole.

```

plotpw := proc(x::symbol)
local a, b, k;
if nargs < 2 then NULL else
for k from 2 to nargs do
if type(args[k], 'list') and type(args[k][2..3], 'list('realcons')) and
`if` (nops(args[k]) = 3, true, map(type, {op(args[k][4..-1])}, 'plotopt')[1])
then a := k elif type(args[k], 'equation') and type(args[k], 'plotopt') then b := k
else error "%1-th argument <%2> is invalid", k, args[k]
end if
end do;
plots[display]([seq(plot(args[k][1], args[1] = args[k][2] .. args[k][3], `if` (nops(args[k]) = 3, NULL,
op(args[k][4..nops(args[k])])), k = 2 .. a), `if` (type(a, 'symbol'), NULL, args[a + 1 .. b]))
end if
end proc
    
```

Typical examples of the procedures use:

> **plotpw(x, [x^2, 0, sqrt(2), thickness=2, color=black, labelfont=[TIMES,BOLD,14]], [sin(x^3),3,4,**  
**thickness = 2, color = red], [1.5, 5, 6, thickness = 2, color = blue], axesfont = [TIMES, BOLD, 12]);**



**d2plot** - dynamic generating of the function graph of one variable

Call format of the procedure:

**d2plot(F, x, d, p {, popt})**

Formal arguments of the procedure:

- F** – an algebraic expression from independent **x**-variable
- x** – symbol defining name of leading variable of **F**
- d** – a range of *realnum*-type
- p** – expression or list of *realnum*-type
- popt** – (optional) *plot*-options

Description of the procedure:

In a series of problems of mathematical and physical & engineering character it is desirable to have graphic means of representation of dynamics of functional dependence depending on change of its leading variable. For example, dynamics of behaviour of some function **F(x)** depending on change of the **x**-variable on some interval. The given problem in the certain extent is being solved by means of the following procedure *d2plot*.

The procedure call **d2plot(F, x, d, p {, popt})** returns the animated graphic 2D-object representing dynamics of change of algebraic expression **F(x)** on an interval **d** with change by the leading **x**-variable according to **p** argument. As the given parameter there can be a value of *realnum*-type or list of *realnum*-type. In the first case the **p**-argument defines an increment for **x**-variable on the interval **d** whereas in the second case **p** defines the list of points of the interval **d** that determine the fixed intervals of change of **x**-variable. At last, the optional *popt*-argument defines sequence of *plot*-options, controlling by appearance of the animated graphic object. Procedure has a series of interesting appendices, for example, in physical modelling and teaching in mathematically oriented disciplines. However, it is necessary to have in mind, that at use of the large enough number of basic **x**-points (*argument p*) on which analysis of **F(x)**-dynamics should be made, can be demanded the sizeable basic resources of a computer what can make such task practically impracticable.

```

d2plot := proc(F::algebraic, x::symbol, d::range(realnum), p::{realnum, list(realnum)})
local a, b, c, h, g, t;
  assign(h = evalf(d), g = evalf(p)), assign(a = g, t = []);
  if type(p, 'list') and not belong(g, h) then error "4-th argument must be in diapason <%1>", d
  elif not type(g, 'list') then
    if rhs(h) <= lhs(h) + g then error "p-increment <%1> is invalid", p
    else a := sort([lhs(h), seq(lhs(h) + c*g, c = lhs(h) .. floor((rhs(h) - lhs(h))/g)), rhs(h)])
    end if
  end if;
  for b in a do
    try t := [op(t), plot(F(x), x = lhs(h) .. b, `if` (4 < nargs and
      {seq(type(args[c], 'plotopt'), c = 5 .. nargs)} = {true}, op([args[5 .. -1]]), NULL))]
    catch : next
  end try
end do;
  plots[display](t, insequence = true)
end proc

```

Typical example of the procedure use:

> **d2plot(x^2\*sin, x, 0 .. 4\*Pi, 0.05, color = blue, thickness = 2, axesfont = [TIMES, BOLD, 14]);**

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

# Chapter 6.

## Software of working with *Maple* datafiles and documents

Being the programming language in the package environment, oriented, first, to symbolic calculations (*computer algebra*) the *Maple* language has the relatively limited opportunities at operation with data that are located in external computer memory. Moreover, in this respect the *Maple* language essentially yields to traditional programming languages such as *C*, *COBOL*, *FORTRAN*, *PL/1*, *Pascal*, *ADA*, *Basic*, etc. At the same time, the *Maple* language, oriented, first of all, onto solution of problems of mathematical character, gives a set of tools for access to datafiles which can quite satisfy a broad enough audience of users of physical and mathematical appendices of the package. In the present section, additional means of access to datafiles are represented, essentially extending opportunities of the package in the given direction. Many of them simplify the programming of many problems dealing with access to datafiles of different purpose. With all evidence we can assert, that new package modules **FileTools** and **LibraryTools** have been inspired by a series of our books with which the *Maple* developers have been acquainted. However, our set of the similar procedures is considerably more representative and is focused on wider practical use. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

### 6.1. General purpose software

In the given item, the tools of access to the data which have general purpose and are oriented to the widest use in problems of access to the datafiles and to a work with file system of a computer as a whole are represented. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**fopen1** - opening of datafiles - an extension of Maple procedure *fopen*

Call format of the procedure:

**fopen1(F, m {, t})**

Formal arguments of the procedure:

- F** - a symbol or string defining a datafile or full path to a datafile to be opened
- m** - an opening mode; may be: *READ*, *WRITE*, or *APPEND*
- t** - (*optional*) a datafile type; may be: *TEXT* or *BINARY*

Description of the procedure:

*Maple 9* (as against the previous releases) incorrectly specifies a status of datafile such as *BINARY*, open by function *fopen* in reading mode *READ*, whereas *Maple 6-10* incorrectly specifies status of datafile of any type open by function *fopen* in writing mode *APPEND*. In some cases, the given circumstance can cause incorrect access to datafiles, if it is based on the information of the function

call *iostatus()*. The suggested procedure *fopen1(F, m {, t})*, having the same set of formal arguments, as standard function *fopen*, eliminates the given defects, in addition having a few useful properties. If file *F* is absent, the procedure call *fopen1(F, 'READ')* irrespective of presence and values of the third argument causes erroneous situation with diagnostics "*datafile <%1> does not exist*". Whereas in other cases the absent datafile is created having the required type and with opening in a required mode; in addition, through the global variable *\_fullpathfile* the full path to the created and open datafile is returned. If file *F* is empty, the procedure call *fopen1(F, 'READ')* irrespective of presence and values of the third argument returns the *NULL* value, i.e. *nothing*, with output of the corresponding message; in addition, the datafile remains closed. The successful procedure call *fopen1* similarly to the standard function *fopen* (excepting the specified case) returns the number of logic I/O channel on which the datafile was open. This procedures appear as a rather useful tools when error-free datafiles handling is demanded. Our experience confirms a rather high efficiency of the procedures use at implementation of means dealing with datafiles processing. The examples represented below very well illustrate the told.

```
fopen1 := proc(F::string, symbol, M::symbol)
local a, b, c, f;
global _fullpathfile;
if not member(M, {'APPEND', 'READ', 'WRITE'}) then error "2nd argument is invalid -
  <%1>; file mode must be {APPEND, READ, WRITE}", M
elif nargs = 3 and not member(args[3], {'TEXT', 'BINARY'})
then error "3rd argument is invalid - <%1>; must be {TEXT, BINARY}", args[3]
elif not type(F, 'file') then if M = 'READ' then error "datafile <%1> does not exist", F
  else f := MkDir(F, 1); _fullpathfile := f
  end if
else f := F; unassign('_fullpathfile')
end if;
c := proc(f)
  local d, k;
  d := iolib(13)[4 .. -1]; seq('if (k[2] = f, RETURN(k[1]), NULL), k = d)
  end proc;
if iolib(6, f, 'infinity') = 0 then b := 0; iolib(7, f) else b := 1; iolib(7, f) end if;
if M = 'READ' then
  if b = 0 then WARNING("datafile <%1> is empty and closed", f)
  elif nargs = 3 and args[3] = 'TEXT' or nargs = 2 then null(iolib(2, f), iolib(6, f, 0)), c(f)
  else null(iolib(4, f), iolib(6, f, 0)), c(f)
  end if
elif M = 'APPEND' then
  if nargs = 3 and args[3] = 'TEXT' or nargs = 2 then null(iolib(5, f, 'if (b = 0, "",
    convert(iolib(4, f), 'bytes')), null(iolib(6, f, 0))), null(iolib(6, f, 'infinity'))), c(f)
  else null(iolib(5, f, null(iolib(6, f, iolib(6, f, null(iolib(6, f, assign('a' = iolib(4, f,
    null(iolib(6, f, iolib(6, f, 'infinity') - b))), 'infinity')), 'infinity') - b)), 'if (b = 0, [], a))), c(f)
  end if
else iolib(1, f, args[2 .. -1])
end if
end proc
```

Typical examples of the procedure use:

```
> Close(); fopen("C:/temp/appendix.zip", APPEND): iostatus(); # Maple 6-10
    [1, 0, 7, [0, "C:/temp/appendix.zip", STREAM, FP = 2009464032, READ, TEXT]]
> Close(); fopen("C:/temp/appendix.zip", APPEND, BINARY): iostatus();
    [1, 0, 7, [0, "C:/temp/appendix.zip", STREAM, FP = 2009464032, READ, BINARY]]
> Close(); fopen("C:/temp/appendix.zip", READ, BINARY): iostatus(); # Maple 9-10
    [1, 0, 7, [0, "C:/temp/appendix.zip", STREAM, FP = 2009464032, WRITE, BINARY]]
> fopen1("C:/temp/avz/agn/vsv", APPEND), iostatus(), _fullpathfile; # Maple 6-10
    0, [1, 0, 7, [0, "c:\temp\avz\agn\vsv", STREAM, FP = 2009464032, WRITE, TEXT]],
    "c:\temp\avz\agn\vsv"
> Close(); fopen1("C:/temp/aaa_f1t.txt", READ), iostatus();
    0, [1, 0, 7, [0, "C:/temp/aaa_f1t.txt", STREAM, FP = 2009464032, READ, TEXT]]
> Close(); fopen1("C:/temp/appendix.zip", APPEND, TEXT), iostatus();
    0, [1, 0, 7, [0, "C:/temp/appendix.zip", STREAM, FP = 2009464032, WRITE, TEXT]]
> Close(); fopen1("C:/aaa/bbb/ccd/ddd", APPEND, BINARY), iostatus();
    0, [1, 0, 7, [0, "c:\aaa\bbb\ccd\ddd", STREAM, FP = 2009464032, WRITE, BINARY]]
> Close(); fopen("C:/temp/ccd/avz", READ, BINARY), iostatus();
    0, [1, 0, 7, [0, "C:/temp/ccd/avz", STREAM, FP = 2009464032, READ, BINARY]]
> fopen1("C:/temp/RANS/IAN/appendix.zip", READ);
    Error, (in fopen1) datafile <C:/temp/RANS/IAN/appendix.zip> does not exist
```

### IOsave - saving and restoring of states of open datafiles

Call format of the procedure:

**IOsave()**

Formal arguments of the procedure: *no*

Description of the procedure:

In a lot of cases there is a necessity of saving of the current condition of open datafiles of the data with the purpose of its restoring at the required moment of the current session or other session with the package. We shall understand by a state of an open datafile the following: (1) number of logic I/O channel on which the datafile is open, (2) a full way to datafile, (3) mode of opening of datafile, (4) type of datafile, (5) scanned position of datafile. The first four positions are reflected by the information returned by the call of built-in function *iostatus*, whereas the information on the fifth position the built-in function *filepos* of the package gives.

Procedure *IOsave* provides an opportunity of saving of state of all open datafiles of the current session, and subsequent restoring of their former state after closing all files. Thus, restoration is made within distribution of logic I/O channels for open channels and their last scanned positions. The state of all open files is saved in a special global variable *\_IOtabStatus* of *table*-type which is saved in a datafile for the purpose of opportunity of the subsequent restoration. Depending on a state of open datafiles of the current session and presence of datafile with variable *\_IOtabStatus* procedure *IOsave* carries out actions as follows.

<i>_IOtabStatus</i>	<i>Open datafiles</i>	<i>Actions of the procedure</i>
<i>no</i>	<i>no</i>	<i>output of the appropriate warning</i>
<i>no</i>	<i>yes</i>	<i>saves the current state of open datafiles with closing all files of the current session</i>

yes	no	restores a state of previously opened datafiles from datafile with variable <code>_IOtabStatus</code> with removal of this file
yes	yes	saves the current state of open datafiles in a file with preliminary restoration of the previous state of datafiles

Each call of procedure `IOSave()` returns state of open datafiles of the current session of the package in format of the built-in function `iostatus` with output of the appropriate warning. In the practical relation the procedure allows to close at any moment datafiles of the current session with opportunity of the subsequent restoration of their interrupted states anytime or in the current session, or in other sessions of work with the package. The examples below of the procedure appendices evidently enough illustrate the told.

```

IOSave := proc()
local a, b, k, f;
global _IOtabStatus;
type(MkDir, 'libobj'), assign(a = iostatus(), f = cat(_libobj, "/IO$Art16_Kr9$"));
if type(f, 'file') and nops(a) = 3 then read f;
  for k in map(op, [indices(_IOtabStatus)]) do
    OpenLN(k, _IOtabStatus[k][2], _IOtabStatus[k][1], _IOtabStatus[k][3],
      `if`(_IOtabStatus[k][2] = 'open', NULL, _IOtabStatus[k][4]));
    filepos(_IOtabStatus[k][1], _IOtabStatus[k][5])
  end do;
iostatus(), remove(f), unassign('_IOtabStatus'),
  WARNING("previous datafile I/O state has been restored")
elif type(f, 'file') and nops(a) <> 3 then read f;
assign('b' = eval(_IOtabStatus)), unassign('_IOtabStatus');
for k from 4 to nops(a) do _IOtabStatus[a[k][1]] := [a[k][2], `if` (a[k][3] = 'RAW', 'open', 'fopen'),
  a[k][5], a[k][6], filepos(a[k][2])]
end do;
save _IOtabStatus, f; Close(), assign('_IOtabStatus' = eval(b));
for k in map(op, [indices(_IOtabStatus)]) do
  OpenLN(k, _IOtabStatus[k][2], _IOtabStatus[k][1], _IOtabStatus[k][3],
    `if`(_IOtabStatus[k][2] = 'open', NULL, _IOtabStatus[k][4]));
  filepos(_IOtabStatus[k][1], _IOtabStatus[k][5])
end do;
iostatus(), unassign('_IOtabStatus'), WARNING("current datafiles I/O state has been saved
  and replaced by previous I/O state")
elif not type(f, 'file') and nops(a) <> 3 then
  for k from 4 to nops(a) do _IOtabStatus[a[k][1]] := [a[k][2], `if` (a[k][3] = 'RAW', 'open', 'fopen'),
    a[k][5], a[k][6], filepos(a[k][2])]
  end do;
  save _IOtabStatus, f; Close(), iostatus(), unassign('_IOtabStatus'), WARNING("current
    datafiles I/O state has been saved; the current session does not contain open datafiles")
else iostatus(), WARNING("the current session does not contain open datafiles and it has not
    the saved I/O state")
end if
end proc

```

Typical examples of the procedure use:

> **IOsave();**

Warning, the current session does not contain open datafiles and it has not the saved I/O state

> **fopen("C:/temp/aaa", READ, BINARY), open("C:/temp/bbb", WRITE), open("C:/temp/ccc", WRITE), fopen("C:/temp/binom.doc", READ, TEXT), iostatus();**

```
0, 1, 2, 3, [4, 0, 7, [0, "C:/temp/aaa", STREAM, FP = 2009464032, READ, BINARY],
[1, "C:/temp/bbb", RAW, FD = 12, WRITE, BINARY],
[2, "C:/temp/ccc", RAW, FD = 13, WRITE, BINARY],
[3, "C:/temp/binom.doc", STREAM, FP = 2009464064, READ, TEXT]]
```

> **IOsave();** ⇒ [0, 0, 7]

Warning, current datafiles I/O state has been saved; current session does not contain open datafiles

> **IOsave();**

Warning, previous datafile I/O state has been restored

```
[4, 0, 7, [0, "C:/temp/aaa", STREAM, FP = 2009464032, READ, BINARY],
[1, "C:/temp/bbb", RAW, FD = 12, WRITE, BINARY],
[2, "C:/temp/ccc", RAW, FD = 13, WRITE, BINARY],
[3, "C:/temp/binom.doc", STREAM, FP = 2009464064, READ, TEXT]]
```

**MkDir** - creation of a directory of any level of nesting and/or a datafile

**MkDir1**

**MkDir2**

**MkDir3**

Call format of the procedures:

**MkDir**(*dirName* {, 1})

**MkDir1**(*d*::{string, symbol})

**MkDir2**(*d*::{string, symbol})

**MkDir3**(*d*::{string, symbol})

Formal arguments of the procedures:

*dirName*, *d* - a name, a chain of directories or a path to create (may be *symbol* or *string*)

**1** - (*optional*) indicator of a datafile creation

Descriptions of the procedures:

The **MkDir** procedure creates a directory, a directories chain and/or a file in file system of the underlying operating system. The *dirName* argument that should be the *Maple* string or symbol specifies a directories chain that should be created. The procedure admits a coding of the actual *dirName* argument by the string or symbol of the following view:

<disk unit:\\Directory/subdirectory\_1\\.../subdirectory\_n>

The set of characters that are permitted in the directories names is system-dependent. Likewise, a character, used to separate the components of a directories chain is system-dependent. If the *backslash* character appears in the string, it must be doubled up, because the *Maple* strings use the *backslash* character as *escape* character.

If the at procedure call **MkDir**(*dirName*, **1**) the second actual argument **1** has been coded, then the procedure considers the last element of a chain defined by the first actual *dirName* argument as a file which is created as the *empty* closed datafile. The created file is located in the last subdirectory of the created directories chain. For example, the procedure call

**MkDir("C:/Temp/Dir/Test147.txt", 1)**

creates both the directories chain "C:/Temp/Dir" (*within missing components*) and the empty closed datafile "C:/Temp/Dir/Test147.txt".

It is necessary to note the following circumstance. The file conception of operating MS DOS system does not distinguish a file itself and a directory. Therefore, even the MS DOS command *mkdir* cannot create subdirectory in a directory containing file of the same name. For elimination of the given lack, the procedure *MkDir* uses an artificial reception consisting in the following. If at creation of the required subdirectory, the procedure finds out presence of file of the same name in the catalogue of previous level the created subdirectory receives the new name formed of the given name by means of adding to it of prefix "\_". Analogous situation takes place if creation of a required file *MkDir* finds out presence of subdirectory of the same name; about that, the appropriate warnings are output. In general case this warning has the following view "Path element <%1> has been found, it has been renamed on <%2>".

A successful call of the *MkDir(F)* procedure keeps the current directory and returns the full path to the required directory or file given by the actual *F* argument. The procedure call *MkDir({'`|'})* returns the *NULL* value only. In addition, if the required path is absent, it is being created with output of an appropriate warning of the above view if it is necessary. An unsuccessful call causes an exception. In contrast to the standard procedure *mkdir* the successful call *MkDir* always returns the required full path. The returned path has the standard *Maple* notation in lowercase. Such approach allows to simplify programming of many problems dealing with access to datafiles. The *MkDir* procedure essentially extends opportunities of the built-in *mkdir* function that allows to create the directories of one level of nesting only. The procedure in a lot cases allows to simplify programming of problems of processing of elements of file system of a computer and problems dealing with access to datafiles of different nature.

Two procedures *MkDir1(d)* and *MkDir2(d)*, creating a chain of directories determined by *d*-argument, have the essentially more limited opportunities relative to *MkDir* procedure, however are much more reactive at creation of chains of directories in "pure" conditions when potential especial situations are reduced to a minimum. The procedure call *MkDir1(d)* returns a full path to the created directory *d*; in addition, if on a path of creation of chain *d* there are files of the same names with directories, the directories corresponding to them will be ended the symbol "\$". Whereas the procedure *MkDir2(d)* supposes absence of the above similar situation and extremely quickly creates the required chain of directories *d*, returning *NULL* value. At last, the procedure *MkDir3(d)* is equivalent to the previous procedure *MkDir2(d)*, however it does not use means of our library.

```
MkDir := proc(F::{string, symbol})
local cd, r, k, h, z, K, L, Lambda, t, d, omega, u, f, s, g, v;
  s := "Path element <%1> has been found, it has been renamed on <%2>";
  if Empty(F) then return NULL
  elif type(F, 'dir') and nargs = 1 then return CF(F)
  elif type(F, 'file') and nargs = 2 then return CF(F)
  else cd := currentdir()
  end if;
  u, K := interface(warnlevel), CF(F);
  assign(Lambda = ((x) -> close(open(x, 'WRITE')))), assign(L = CFF(K), omega = 0);
  assign(r = cat(L[1], "\\\"), `if` (nargs = 2 and args[2] = 1,
    assign('L' = L[1 .. -2], 'omega' = L[-1], t = 1), 1);
  if L = [] then assign(g = cat(r, r[1]), v = cat(r, "_", r[1]));
  if t <> 1 then return r
  elif type(g, 'dir') then return null(Lambda(v)), v
  elif type(g, 'file') then return g
```

```

else return null(Lambda(g)), g
end if
end if;
for k from 2 to nops(L) do currentdir(r), assign('g' = cat(r, L[k]));
if type(g, 'dir') then
try mkdir(g) catch: NULL end try
elif type(g, 'file') then assign('L'[k] = cat("_", L[k]), 'd' = 9);
try mkdir(cat(r, L[k]))
catch "file I/O error": d := 9
catch "directory exists and is not empty": d := 9
end try;
assign('d' = 9), WARNING(s, L[k][2 .. -1], L[k])
else mkdir(cat(r, L[k]))
end if;
assign('r' = cat(r, L[k], "\\"))
end do;
if t = 1 then
if type(cat(r, omega), 'dir') then op([Lambda(cat(r, "_", omega)),
WARNING(s, omega, cat("_", omega))])
else return `if` (d = 9, cat(r, omega), CF(F)),
Lambda(cat(r, omega)), null(currentdir(cd))
end if;
cat(r, "_", omega), null(currentdir(cd))
else null(currentdir(cd)), `if` (d = 9, r[1 .. -2], CF(F))
end if
end proc
MkDir1 := proc(d::{string, symbol})
local a, b, c;
assign(b = CFF(d)), assign(c = b[1]);
for a from 2 to nops(b) do c := cat(c, "\\ ", b[a]);
try mkdir(c)
catch "directory exists and is not empty":
if type(c, 'file') then
try c := cat(c, "$"); mkdir(c) catch "directory exists and is not empty": next end try
end if;
next
end try
end do;
c
end proc
MkDir2 := proc(d::{string, symbol})
local a, b, c;
assign(b = CFF(d)), assign(c = b[1]);
for a from 2 to nops(b) do c := cat(c, "\\ ", b[a]);
try mkdir(c) catch "directory exists and is not empty": next end try

```

```

end do
end proc
MkDir3 := proc(L::{string, symbol})
local a, b, c;
  assign(a = "", b = ""), [seq(^if (c = "\\\" or c = "/" , assign('a' = cat(a, "", "")), assign('a' = cat(a, c))),
    c = cat("", L))];
  for c in parse(cat("[", a, ""])) do b := cat(b, c, "\\");
    try mkdir(b)
    catch "directory exists and is not empty": next
    catch "file I/O error": next
    catch "permission denied": next
  end try
end do
end proc

```

Typical examples of the procedures use:

```

> MkDir("C:/Temp/rans/ian\\VGTU/AGN\\Art/Kr\\Svet/Arne");
      "c:\temp\rans\ian\vgtu\agn\art\kr\svet\arne"
> MkDir("C:/Temp/rans/ian\\VTU/AGN\\Art/Kr\\Svet/Arne\\RANS.ian", 1);
      "c:\temp\rans\ian\vtu\agn\art\kr\svet\arne\rans.ian"
> MkDir("C:/Temp/rans/ian\\VTU/AGN\\Art/Kr\\Svet/Arne\\RANS.ian");
      Warning, Path element <rans.ian> has been found, it has been renamed on _rans.ian
      "c:\temp\rans\ian\vtu\agn\art\kr\svet\arne\_rans.ian"
> t:= time(): MkDir1("C:\\1/2/3\\4/5/6/7\\8\\9/10/11/12\\13/14\\15\\16"), time() - t;
      "c:\1\2\3\4\5\6\7\8\9\10\11\12\13\14\15\16", 0.141
> t:= time(): MkDir2("C:\\1/2/3/4/5/6/7/8\\9\\10/11/12/13/14\\15\\16"), time() - t; ⇒ 0.0
> MkDir3("D:\\RANS\\IAN\\RAC\\REA\\Tallinn\\Vilnius\\Grodno");

```

## 6.2. Software for operation with BINARY datafiles

As against the software of the previous item providing access to external files of *Maple* at a level of logical records (lines, sequences of *Maple* expressions, the whole *Maple* clauses), tools represented below, provide access to datafiles at a level of separate symbols (*bytes*). For datafiles processing of this type, the package *Maple* has appropriate built-in functions of access to *BINARY* datafiles. The procedures represented below extend tools of the package at processing of *BINARY* datafiles by allowing to solve a number of problems dealing with *BINARY* datafiles the more effectively. Some of them have made a good showing from the viewpoint of practical programming. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**Fend** – handling of especial situation “the end of a datafile”

**Find**

Call format of the procedure:

**Fend**(F {, 'h'})

**Find**(n)

Formal arguments of the procedures:

- F** - string or symbol defining a *file qualifier (filename or full path)*, or an integer from range 0 .. 6
- h** - (*optional*) an assignable name
- n** - number of logical I/O channel (*integer from range 0 .. 6*)

Descriptions of the procedures:

Essential distinctions between datafiles of types *TEXT* and *BINARY* are observed relative to handling of an especial situation "*the end of a datafile*". For *BINARY* datafiles, *Maple* has not tools of handling of the specified situation and in each concrete case this problem of handling lays on the user, what not always is desirable. The given question is considered in our books [8-14,29-33,43,44] enough in detail. With the purpose of solution of the above problem, the *Fend* procedure has been introduced.

Extremely useful procedure *Fend* provides handling of the situation "*the end of a datafile*" for datafiles of any type. The procedure call *Fend(F {, 'h'})* returns the *true* value in case of detection of situation "*the end of a datafile*" and the *false* value otherwise for a datafile specified by the first actual **F** argument (qualifier or number of a logical I/O channel). In case of coding of the second optional **h** argument (in case of the *false* value) via it the list of view [*scanned position of a file, length of the rest of a file*] is returned. The *Fend* procedure makes sense only for an open datafile and does handling of the basic especial and erroneous situations. The durable experience of usage of the given procedure has shown its quite satisfactory operating characteristics. The procedure generates erratic situations "*file descriptor <%1> is invalid*", "*<%1> is not a file*", "*all datafiles are closed*" and "*<%1>: file is closed or file descriptor is not used*" whose sense is quite clear. As against *Fend*, the procedure *Find(n)* supposes only one actual **n** argument as that there can be only number of the logic I/O channel, ascribed to an open datafile.

```
Fend := proc(F::{string, symbol, integer})
local k, a, b, c, p, h;
  if type(F, 'integer') then
    if not member(F, {k $ (k = 0 .. 6)}) then error "datafile descriptor <%1> is invalid", F
    else a := iostatus()
    end if
  elif type(F, 'file') then a := iostatus()
  else error "datafile <%1> does not exist", F
  end if;
  if a = [0, 0, 7] then error "all datafiles are closed"
  else seq(`if` (a[k][1] = F or a[k][2] = cat("", F), assign('h' = 9, 'c' = a[k][1]), NULL), k = 4 .. nops(a))
  end if;
  if h <> 9 then error "<%1>: datafile is closed or datafile descriptor is not used", F end if;
  try
    assign(p = filepos(c));
    if filepos(c, `infinity`) <= p then return assign(b = filepos(c, p)), `true`
    else return `false`, `if` (nargs = 1, assign(b = filepos(c, p)),
      op([assign([args][2] = [p, filepos(c, `infinity`) - p]), assign(b = filepos(c, p))]))
    end if
  catch: null("Processing of an especial situation with datafiles {direct, process, pipe}")
  end try
end proc
Fend := proc(F::{string, symbol, integer})
```

```

local k, a, b, c, p, h;
  assign(a = iostatus()), seq(`if` (a[k][1] = F or a[k][2] = cat("", F),
    assign('h' = 9, 'c' = a[k][1], NULL), k = 4 .. nops(a));
if h <> 9 then error "<%1>: datafile is closed or datafile descriptor is not used", F end if;
try
  assign(p = filepos(c));
if filepos(c, infinity) <= p then return assign(b = filepos(c, p)), true
else return false, `if` (nargs = 1, op([NULL, assign(b = filepos(c, p))]),
  op([assign([args][2] = [p, filepos(c, infinity) - p]), assign(b = filepos(c, p))]))
end if
catch: null("Processing of an especial situation with datafiles {direct, process, pipe}")
end try
end proc
Find := (F::{0, 1, 2, 3, 4, 5, 6}) -> [ `if` (type(eval(cat(__filesize, F)), 'symbol'),
  [assign(cat(__filesize, F) = filepos(F, infinity)), filepos(F, 0)], NULL),
  `if` (eval(cat(__filesize, F)) <= filepos(F), [true, unassign(cat(__filesize, F))][1], false)] [-1]

```

Typical examples of the procedures use:

```
> filepos("C:/RANS/ABS.txt", infinity), Fend("C:/RANS/ABS.txt", 'h'), h;
          917, true, [10]
```

```
> filepos("C:/RANS/ABS.txt", 61), Fend("C:/RANS/ABS.txt", 'h'), h;
          61, false, [61, 856]
```

```
> F:=fopen("C:/rans/Simona.txt", READ, BINARY): while not Fend(F) do h:=readbytes(F):
writebytes("C:/Temp/Simona.txt", h) end do: Close(); Fequal("C:/RANS/Simona.txt",
"C:/Temp/Simona.txt");
```

```

          true
> Fend("C:/Temp/Demo/Demo/DemoLib/Maple.hdb");
```

Error, (in Fend) <C:/Program Files/Maple 9/Intro.mws>: file is closed or file descriptor is not used

### 6.3. Software for operation with *Maple* files

The given item represents a number of procedures providing a few additional possibilities at operation with *Maple* files, i.e. files whose names have expansions {*mws*, *m*, *mw*} or can be successfully read by means of the statement **read**. Files of the given kinds occupy an especial place in the *Maple* file system. The *mws*-files document a job of the user in the *Maple* environment, whereas the *m*-files form elements of both the *Maple* libraries and the user libraries. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

#### **hidemws** - hiding of *Maple* files

Call format of the procedure:

```
hidemws(F::{symbol, string})
```

Formal arguments of the procedure:

F - filename or full path to a file of type {".m", ".mws"}

Description of the procedure:

Worksheet *Maple* files are identified by names ending with the three characters ".mws". Such *Maple* files are portable between the graphical user interfaces on different operating platforms. *Maple* internal format files are used to store procedures and other objects in a more compact format. *Maple*

internal format files are identified by file names ending with the two characters ".m" (or ".M" on operating platforms where file names are not case sensitive). In a series of cases it is desirable to hide the *Maple* files of type {".m", ".mws"} from loading into the current session.

After procedure call *hidemws(F)*, the *Maple* file *F* of type {".m", ".mws"} will be inaccessible for loading into the current session, and vice versa; i.e. the procedure call works according to the switch principle. The procedure call returns the *NULL* value, i.e. nothing. In addition, the hidden *mws*-file is not being loaded into the current session without output of any warning, whereas an attempt to load the hidden *m*-file by means of the **read** statement causes erroneous situation. The posterior procedure call *hidemws(F)* makes the hidden file *F* available for loading into the current session, and vice versa.

```

hidemws := proc(F::{string, symbol})
local a, b, c;
  c := x -> [x[8], x[2], x[6], (x[4] + 1) mod 2, (x[5] + 1) mod 2, x[3], x[7], x[1]];
  try open(F, 'READ'), close(F)
  catch "file or directory does not exist": error "datafile <%1> does not exist", F
  end try;
  if cat(" ", F)[-4 .. -1] = ".mws" then
    filepos(F, 11), assign(a = readbytes(F)), filepos(F, 11), `if` (a = [48], writebytes(F, [79]),
    writebytes(F, [48])); close(F)
  elif member(cat(" ", F)[-2 .. -1], {".m", ".M"}) then
    assign(b = filepos(F, `infinity`)), filepos(F, b - 3), assign(a = readbytes(F)), filepos(F, b - 3),
    writebytes(F, map(Bit1, a, c)); close(F)
  end if
end proc

```

Typical examples of the procedure use:

```

> hidemws("C:\\temp\\svegal\\Art_Kr\\uplib.mws");
> hidemws("C:\\temp\\svegal\\Art_Kr\\hhh.m");
> read("C:\\temp\\test.m");
  Error, at offset 137 in `C:/temp/test.m`, unexpected DAG type: 0, 122, (z)
> hidemws("C:\\temp\\test.m");
> read("C:\\temp\\test.m");

```

**rtfnin** - deleting of input-paragraphs out of *rtf*-files

Call format of the procedure:

**rtfnin(F, O)**

Formal arguments of the procedure:

- F** – symbol or string defining a filename or full path to *rtf*-файлы
- O** – symbol or string defining a target *rtf*-file

Description of the procedure:

In some cases there is a necessity to delete *Input*-paragraphs out of the *Maple* document saved or exported in *rtf*-format. *Maple* has no standard means for the decision of this problem whereas simple procedure *rtfnin* solves this problem.

The procedure call *rtfnin(F, O)* returns the full path to target *rtf*-file **O** that differs from source *rtf*-file **F** by absence of *Input*-paragraphs. If the source *rtf*-file not contain *Input*-paragraphs, the procedure call *rtfnin(F, O)* returns the *NULL* value with output of appropriate warning; in addition,

target *rtf*-file **O** is created being a copy of the source *rtf*-file. At absence of path to the target *rtf*-file the procedure creates path to its analog with output of appropriate warning. Such possibility is provided by procedure *pathtf*.

```

rtfnin := proc(F::file, O::{string, symbol})
local a, b, c, d, f, g, h, p, t, x, y;
  if Ftype(F) = ".rtf" then assign(a = "\\pard\\plain", b = "\\par", f = pathtf(O),
    g = "\\additive Maple Input"); close(F);
  else error "source datafile is not a rtf-file"
  end if;
  do
    c := readline(F);
    if c = 0 then break
    elif search(c, g) then search(c, "\\b", 't'); h := cat("{", c[4 .. t + 1], " > "); t := 63; next
    elif c <> a then writeline(f, c); next
    else x := readline(F); y := readline(F)
    end if;
    if y = "> " or Suffix(y, h, p) then
      do if readline(F) = b then break end if end do
    else writeline(f, c), writeline(f, x), writeline(f, y);
      do
        d := readline(F);
        if d = b then writeline(f, b); break
        else writeline(f, d)
        end if
      end do
    end if
  end do;
  close(F, f), `if` (t = 63, f, WARNING("source rtf-file does not contain Input Maple paragraphs"))
end proc

```

Typical examples of the procedure use:

```
> rtfnin("C:\\temp\\aaa\\bbb\\ccc\\aaa9c.rtf", "C:\\temp\\aaa9c1.rtf");
```

Warning, source rtf-file does not contain Input Maple paragraphs

```
> rtfnin("C:\\temp\\reprolib.rtf", "C:\\temp\\reprolib1.rtf");
```

"c:\temp\reprolib1.rtf"

```
> rtfnin("C:\\temp\\aaa8.rtf", "C:\\temp\\aaa\\bbb\\ccc\\rtf67.rtf\\svetla.rtf");
```

Warning, target path <c:\temp\aaa\bbb\ccc\\_rtf67.rtf\svetla.rtf> has been activated instead of the required path <c:\temp\aaa\bbb\ccc\rtf67.rtf\svetla.rtf>

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259].

# Chapter 7.

## Certain useful tools for work in Maple

The chapter presents certain means useful for work in Maple. The full set of means of the given type is presented in our books [239, 240] and in the library [259] attached to the present book.

**braces** – *output of systems of equations and/or inequalities in conventionalized view*

Call format of the procedure:

**braces(L::anything)**

Formal arguments of the procedure:

**L** – a list or a set of equations and/or inequalities, or function

Description of the procedure:

The useful procedure **braces(L)** provides output of systems of equations and/or inequalities in conventionalized view. Feature of such presentation will be, that numbering of the equations of system is situated at the left, instead of to the right of them. Procedure call **braces(L::{list(relation), set(relation)})** returns the system of equations and/or inequalities **L** given as a set or list in the above view. Whereas the procedure call **braces(L::function)** returns the list of equations and/or inequalities represented in the above view (*if L is piecewise function, the calls braces(L) and convert(L, list) are equivalent*); otherwise, the procedure call **braces(L)** returns the **NULL** value, i.e. nothing.

```
braces := proc(L::anything)
local a, f, k, t;
global __vsvak38;
  if whattype(L) = 'function' then
    try assign(a = convert(L, 'list')), assign(f = [seq(a[k], k = {seq(2*t - 1, t = 1 .. 1/2*nops(a)})]),
      `if (nops(f) = 0, NULL, f)
    catch "": NULL
    end try
  elif type(L, {'list'('relation'), 'set'('relation')})
  then assign(a = "__vsv38ak:=piecewise(", f = cat(currentdir(), "/__$$$__"));
    seq(assign('a' = cat(a, convert(L[k], 'string'), ",", convert([k], 'string'), ",")), k = 1 .. nops(L));
    writeline(f, cat(a[1 .. -2], ");")); close(f);
    (proc() read f end proc)(f), remove(f), __vsv38ak, unassign('__vsv38ak')
  else
  end if
end proc
braces := proc(L::anything)
local a, f, k, t;
  if whattype(L) = 'function' then
```

```

try assign(a = convert(L, 'list')), assign(f = [seq(a[k], k = {seq(2*t - 1, t = 1 .. 1/2*nops(a))})]),
  if(nops(f) = 0, NULL, f)
catch "": NULL
end try
elif type(L, {'list'('relation'), 'set'('relation')}) then assign(a = "piecewise(");
  seq(assign('a' = cat(a, convert(L[k], 'string'), ",", convert([k], 'string'), ",")), k = 1 .. nops(L));
  eval(parse(cat(a[1 .. -2], ")))
else
end if
end proc

```

Typical examples of the procedure use:

```
> braces({x^2+a*x+c <= 9, x*cos(y)+sqrt(x*y) <> 16, sqrt(x+y)-z/(x+y)=0, sin(x)+cos(y) >= 3});
```

$$\begin{cases}
 [1] & 3 \leq \sin(x) + \cos(y) \\
 [2] & x \cos(y) + \sqrt{xy} \neq 16 \\
 [3] & \sqrt{x+y} - \frac{z}{x+y} = 0 \\
 [4] & x^2 + ax + c \leq 9
 \end{cases}$$

```
> braces(%);
```

$$\left[ 3 \leq \sin(x) + \cos(y), x \cos(y) + \sqrt{xy} \neq 16, \sqrt{x+y} - \frac{z}{x+y} = 0, x^2 + ax + c \leq 9 \right]$$

```
> R:=braces([a*diff(f(t),t$2)+b*diff(h(t),t)+c*sin(t)=0,x*diff(f(t),t$2)+sin(t)*diff(h(t),t)+exp(t)=0]);
```

$$R := \begin{cases}
 [1] & a \left( \frac{d^2}{dt^2} f(t) \right) + b \left( \frac{d}{dt} h(t) \right) + c \sin(t) = 0 \\
 [2] & x \left( \frac{d^2}{dt^2} f(t) \right) + \sin(t) \left( \frac{d}{dt} h(t) \right) + e^t = 0
 \end{cases}$$

```
> braces(R);
```

$$\left[ a \left( \frac{d^2}{dt^2} f(t) \right) + b \left( \frac{d}{dt} h(t) \right) + c \sin(t) = 0, x \left( \frac{d^2}{dt^2} f(t) \right) + \sin(t) \left( \frac{d}{dt} h(t) \right) + e^t = 0 \right]$$

**diff - the generalized differentiation or partial differentiation**

Call format of the procedure:

```
diff(A {, x {, y {, z ... } })
```

Formal arguments of the procedure:

- A - algebraic expression
- x, y, z, ... - names and/or algebraic expressions

Description of the procedure:

In a whole series of cases the standard procedure *diff(A)* does not allow to calculate correctly both usual and partial derivatives when subexpressions, for example, functions are used as variables of differentiation of an algebraic expression **A**. The following procedure *diff* allows to solve this problem. The procedure call has one or more actual arguments. The first argument defines a differentiated algebraic expression **A** whereas the others define the generalized variables over which differentiation of **A** expression is made. As the generalized variables any algebraic

expressions, admitted *Maple* can act. The procedure call **diff(A {, x {, y {, z ... } })** is equivalent to the call **diff(A {, x {, y {, z ... } })**, where {x, y, z, ...} are usual variables, i.e. for which *map(type, {x, y, z, ...}, symbol) -> {true}*. In addition, the procedure call **diff(A)** returns **A**. *diff* computes the partial derivative of the expression **A** with respect to **x, y, z, ...**, respectively. Procedure handles the basic erroneous and especial situations. In the rest, the description of *diff* corresponds to the description of standard procedure *diff*.

```
diff := proc(A::algebraic)
local a, b, c, d, k, n, Algsubs;
Algsubs := proc(S::list({`=`, 'equation'}), set({`=`, 'equation'}), A::anything)
local a, k;
assign(a = A), seq(assign('a' = algsubs(k, a, 'exact')), k = S), a
end proc;
if nargs = 1 then A else
assign(a = [], b = []);
seq('if (type(k, 'symbol'), assign('a' = [op(a), k]), assign('b' = [op(b), k])), k = [args[2 .. -1]));
if b = [] then diff(args) else
seq('if (whattype(k) = `+`, RETURN(A, WARNING("derivative cannot be evaluated
over integration variable %1", k)), NULL), k = b);
assign(c = {op(b)}), assign(n = nops(c)), assign('d' = {seq(c[k] = cat(_, k), k = 1 .. n)});
subs({seq(rhs(d[k]) = lhs(d[k]), k = 1 .. n)}, diff(Algsubs(d, A), op(subs(d, [args[2 .. -1]])));
if % = 0 then %, WARNING("zero value can be conditioned by inadequacy of substitutions")
else %
end if
end if
end if
end proc
```

Typical examples of the procedure use:

> **A := (a\*cos(f(t)) + b\*sin(ln(h(t)))/(f(t) + h(t)) - c\*exp(f(t) - h(t))); diff(A, f(t), h(t), h(t)\$2);**

$$A := \frac{a \cos(f(t)) + b \sin(\ln(h(t)))}{f(t) + h(t)} - c e^{(f(t) - h(t))}$$

$$\frac{6 a \sin(f(t))}{(f(t) + h(t))^4} - \frac{b \cos(\ln(h(t)))}{h(t)^3 (f(t) + h(t))^2} - \frac{3 b \sin(\ln(h(t)))}{h(t)^3 (f(t) + h(t))^2} - \frac{6 b \sin(\ln(h(t)))}{h(t)^2 (f(t) + h(t))^3} - \frac{6 b \cos(\ln(h(t)))}{h(t)^2 (f(t) + h(t))^3} - \frac{18 b \cos(\ln(h(t)))}{h(t) (f(t) + h(t))^4}$$

$$+ \frac{24 (a \cos(f(t)) + b \sin(\ln(h(t))))}{(f(t) + h(t))^5} + c e^{(f(t) - h(t))}$$

**intt - the generalized multiple indefinite integration**

Call format of the procedure:

**intt(A {, x {, y {, z, ... } })**

Formal arguments of the procedure:

- A** - algebraic expression, the integrand
- x, y, z, ...** - names and/or algebraic expressions

Description of the procedure:

In a whole series of cases the standard procedure *int(A)* does not allow to calculate correctly both usual and multiple indefinite integrals when subexpressions, for example, functions are used as integration variables of an algebraic expression **A**. The following procedure *intt* allows to solve this problem. The procedure call has one or more actual arguments. The first argument defines an

algebraic expression **A** (*the integrand*) whereas the others define the generalized variables over which integration of **A** expression is made. As the generalized variables any algebraic expressions, admitted *Maple* can act. The procedure call **intt(A {, x {, y {, z, ... } })** is equivalent to the call **int(int ... (int(A, x) ... , y), z) ... )**, where {x, y, z, ... } are generalized variables. In addition, the procedure call **intt(A)** returns **A**. **intt** computes both usual and multiple indefinite integrals of the expression **A** with respect to **x, y, z, ...**, respectively. Procedure handles the basic erroneous and especial situations. Actual arguments of the procedure, since the second, are coded similarly to a case of the above procedure **diff ( diff)**.

```

intt := proc(A::algebraic)
local a, b, c, d, k, n, Algsubs, i1, i2, r1, r2, p, f;
global _IntRes;
Algsubs := proc(S::list({`=`, 'equation'}), set({`=`, 'equation'}), A::anything)
local a, k;
    assign(a = A), seq(assign('a' = algsubs(k, a, 'exact')), k = S), a
end proc;
if nargs = 1 then A
else assign(a = [], b = [], r1 = "", r2 = "", f = cat(currentdir(), "\\_svegal_"));
    seq('if (type(k, 'symbol'), assign('a' = [op(a), k]), assign('b' = [op(b), k])), k = [args[2 .. -1]]);
if b <> [] then
    seq('if (whattype(k) = `+`, RETURN(A, WARNING("integration cannot be done over
        integration variable %1", k)), NULL), k = b);
    assign(c = {op(b)}), assign(n = nops(c)), assign('d' = {seq(c[k] = cat(_, k), k = 1 .. n)});
    assign(i1 = Algsubs(d, A), i2 = subs(d, [args[2 .. -1]]), assign(p = nops(i2))
else assign(i1 = A, i2 = [args[2 .. -1]]), assign(p = nops(i2))
end if;
    seq(assign('r1' = cat(r1, "int("), k = 1 .. p), assign('r1' = cat(r1, convert(i1, 'string'), ", "));
    seq(assign('r2' = cat(r2, cat(convert(i2[k], 'string'), ", "))), k = 1 .. p);
    assign('r1' = cat("_IntRes:=", r1, r2[1 .. -2], ":"), writeline(f, r1), close(f);
read f;
if b <> [] then subs({seq(rhs(d[k]) = lhs(d[k]), k = 1 .. n)}, _IntRes),
    unassign('_IntRes'), remove(f) else _IntRes, unassign('_IntRes'), remove(f) end if
end if
end proc

```

Typical examples of the procedure use:

> **A:=(a\*cos(f(t))+b\*sin(ln(h(t)))/(f(t)+h(t))-c\*exp(f(t)-h(t))): intt(A, f(t), h(t)\$2); intt(A, a, b, h(t));**

$$\int \int a \operatorname{Si}(f(t) + h(t)) \sin(h(t)) + a \operatorname{Ci}(f(t) + h(t)) \cos(h(t)) + b \sin(\ln(h(t))) \ln(f(t) + h(t)) - c e^{(f(t) - h(t))} dh(t) dh(t)$$

$$\int \frac{\frac{1}{2} a^2 \cos(f(t)) b + \frac{1}{2} b^2 \sin(\ln(h(t))) a}{f(t) + h(t)} - c e^{(f(t) - h(t))} a b dh(t)$$

**fminimax** – approximation of minimax points for procedures and functions

Call format of the procedure:

```

fminimax(F::{function, procedure}, t::float, x::range(realnum)
    {, y::range(realnum) {, z::range(realnum)}}, {, 'h'})

```

Formal arguments of the procedure:

- F** - valid function or procedure from one, two or three independent variables (**x**, **y**, **z**)
- x** - range for **x**-variable
- y** - (*optional*) range for **y**-variable
- z** - (*optional*) range for **z**-variable
- t** - demanded calculation accuracy
- h** - (*optional*) an assignable name

Description of the procedure:

The procedure *fminimax* is intended for approximation of *minimax* points for a function or procedure from one, two and three independent variables with given accuracy **t**. An exploration array is limited by boundary conditions given by ranges of the form **a .. b**, where **a** and **b** are *realnum*-values. Demanded calculation accuracy is defined as **0.00...0d** (**d** - *posint*); it is being taken into consideration at approximation of *minimax* points. While a procedure or function **F** is defined as follows:

$$F := (x_1, x_2, \dots, x_n) \rightarrow G(x_1, x_2, \dots, x_n), \text{ where } n = \{1 | 2 | 3\}$$

The procedure call *fminimax(F, t, a..b, y=c..d, ...)*, in general, returns the 4-element the nested list whose first element represents the minimal value of **F** over an *realnum*-array defined by ranges **{x, y, z}**, the second represent the point with this value, the third element represent the maximal value of **F**, and the fourth represent the point with this value. If the optional argument **h** is given, through it the table is returned. Meanwhile, the given procedure provides obtaining only of two extreme points, i.e. up to one minimum and one maximum.

In addition, the table entry for index **1** defines quantity of iterations required for evaluation; entry for index **2** defines the set of points in which **F** has singularity, and entry for index **3** defines the set of points in which **F** has complex value. In addition, for case **F(x, y)** and **F(x, y, z)** instead of sets the procedure forms graphic structures which allow to output graphic representations of points of the above sets. These graphs can be obtained on the basis of the following call: *plots[display](h[{2 | 3}])*.

The procedure handles basic especial and erroneous situations. The procedure *fminimax* is slow enough and demands the high-performance PC, however in a lot of cases it allows to estimate minimaxes for functions of complex behaviour when other approaches appear helpless. Examples below illustrate use of the procedure.

```
fminimax := proc(F::{function, procedure}, t::float)
local a, b, c, d, x1, x2, x3, min1, min2, max1, max2, n, h1, h2, h3, h, cv, et, p, a1, a2, a3, b1, b2, b3, G;
  assign(n = nops([rhs(ParProc(F)[1, 1]))]);
  if not member(n, {1, 2, 3}) then error "procedure <%1> should have 1, 2 or 3 arguments", F
  elif not map(type, {args[3 .. n + 2]}, 'range'('realnum')) = {`true`} then
    error "discrepancy of number of arguments of a function and ranges for them" end if;
  G := proc(y) if type(y, 'realnum') then try evalf(y, length(denom(convert(t, 'fraction')) - 1)
    catch : return evalf(y, 1) end try else error "invalid value <%1> for diapason bounds", y
  end if end proc;
  assign(et = {}, cv = {}, d = 0, c = 0, p = 16, h1 = 1/10*G(rhs(args[3]) - lhs(args[3])),
    a1 = G(lhs(args[3])), b1 = G(rhs(args[3])), `if` (n = 1, NULL,
    `if` (n = 2, op([h2 = 1/10*G(rhs(args[4]) - lhs(args[4])), a2 = G(lhs(args[4])),
    b2 = G(rhs(args[4]))]), op([h2 = 1/10*G(rhs(args[4]) - lhs(args[4])), a2 = G(lhs(args[4])),
    b2 = G(rhs(args[4])), h3 = 1/10*G(rhs(args[5]) - lhs(args[5])),
    a3 = G(lhs(args[5])), b3 = G(rhs(args[5])))]));
```

```

do
  if n = 3 then for x1 from a1 by h1 to b1 do for x2
    from a2 by h2 to b2 do for x3 from a3 by h3
      to b3 do h := evalf(F(x1, x2, x3));
        if h = Float('infinity') or h = Float('undefined') then et := {op(et), [x1, x2, x3]}; next
        elif type(h, 'complex1') then cv := {[x1, x2, x3], op(cv)}; next
        else
          if p <> 8 then assign('p' = 8, 'min1' = h, 'a' = [x1, x2, x3], 'max1' = h, 'b' = [x1, x2, x3])
            end if
          end if;
          if h < min1 then min1 := h; a := [x1, x2, x3]
            elif max1 < h then max1 := h; b := [x1, x2, x3]
              end if
            end do
          end do
        end do
      end do
    end do
  elif n = 2 then for x1 from a1 by h1 to b1 do for x2
    from a2 by h2 to b2 do h := evalf(F(x1, x2));
      if h = Float('infinity') or h = Float('undefined') then et := {op(et), [x1, x2]}; next
      elif type(h, 'complex1') then cv := {[x1, x2], op(cv)}; next
      else
        if p <> 8 then assign('p' = 8, 'min1' = h, 'a' = [x1, x2], 'max1' = h, 'b' = [x1, x2]) end if
        end if;
        if h < min1 then min1 := h; a := [x1, x2] elif max1 < h then max1 := h; b := [x1, x2]
          end if
        end do
      end do
    end do
  else for x1 from a1 by h1 to b1 do h := evalf(F(x1));
    if h = Float('infinity') or h = Float('undefined') then et := {op(et), [x1]}; next
    elif type(h, 'complex1') then cv := {[x1], op(cv)}; next
    else
      if p <> 8 then assign('p' = 8, 'min1' = h, 'a' = [x1], 'max1' = h, 'b' = [x1]) end if
      end if;
      if h < min1 then min1 := h; a := [x1] elif max1 < h then max1 := h; b := [x1] end if
    end do
  end if;
  if type(min1, 'symbol') and type(max1, 'symbol') then break end if;
  assign('c' = c + 1, `if` (type(min2, 'symbol'), assign('min2' = 0), 8),
    `if` (type(max2, 'symbol'), assign('max2' = 0), 16);
  if abs(min1 - min2) <= t and abs(max1 - max2) <= t then break
  else assign('d' = d + 1, 'min2' = min1, 'max2' = max1);
    assign('h1' = h1/2^d, `if` (n = 1, NULL, `if` (n = 2, 'h2' = h2/2^d,
      op(['h2' = h2/2^d, 'h3' = h3/2^d])))
  end if
end do;
[ `if` (type(min2, 'symbol'), `minimum is absent`, op([nulldel(min2), map(nulldel, a)])],

```

```

`if` (type(max2, 'symbol'), `maximum is absent`, op([nulldel(max2), map(nulldel, b)])),
`if` (n = 1 and 3 < nargs and type(args[4], 'symbol'),
    assign(args[4] = table([1 = c, 2 = map2(map, nulldel, et), 3 = map2(map, nulldel, cv)])),
`if` (n = 2 and 4 < nargs and type(args[5], 'symbol'),
    assign(args[5] = table([1 = c, 2 = `if` (et = {}, et,
plots[pointplot](map2(map, nulldel, et), 'symbol' = 'DIAMOND', 'symbolsize' = 16,
'axes' = 'NORMAL')), 3 = `if` (cv = {}, cv, plots[pointplot](map2(map, nulldel, cv),
'symbol' = 'DIAMOND', 'symbolsize' = 16, 'axes' = 'NORMAL'))))),
`if` (n = 3 and 5 < nargs and type(args[6], 'symbol'),
    assign(args[6] = table([1 = c, 2 = `if` (et = {}, et,
plots[pointplot3d](map2(map, nulldel, et), 'symbol' = 'DIAMOND', 'symbolsize' = 16,
'axes' = 'NORMAL')), 3 = `if` (cv = {}, cv, plots[pointplot3d](map2(map, nulldel, cv),
'symbol' = 'DIAMOND', 'symbolsize' = 16, 'axes' = 'NORMAL'))))), NULL)))
end proc

```

Typical examples of the procedure use:

> S:=(t, b, u) -> 1/2\*exp(-b\*t)\*(erfc(1/2\*(u - t)/t^(1/2)) + erfc(1/2\*(u + t)/t^(1/2))\*exp(u));

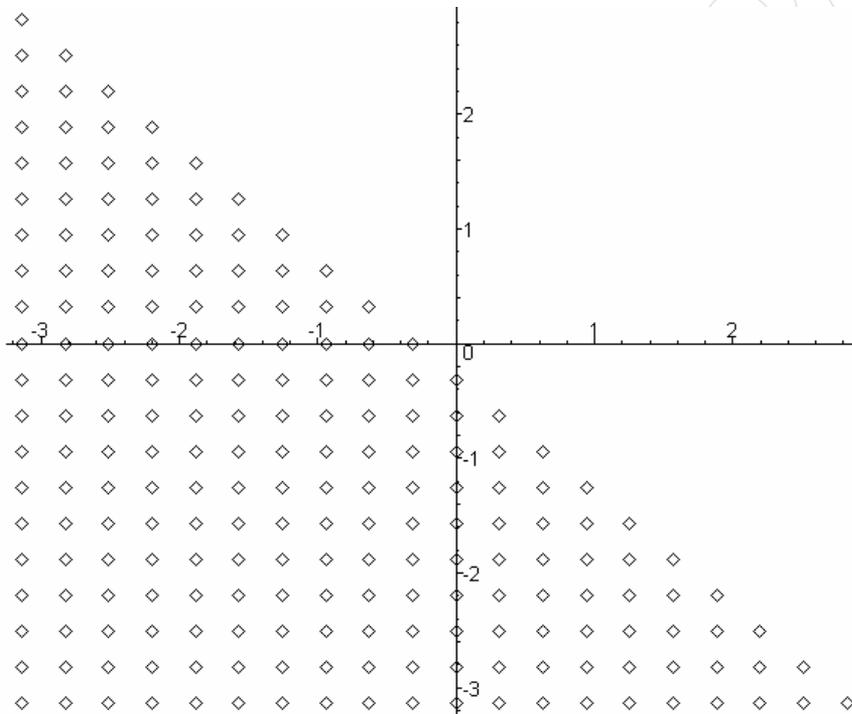
$$S := (t, b, u) \rightarrow \frac{1}{2} e^{-bt} \left( \operatorname{erfc} \left( \frac{1}{2} \frac{u-t}{\sqrt{t}} \right) + \operatorname{erfc} \left( \frac{1}{2} \frac{u+t}{\sqrt{t}} \right) e^u \right)$$

> t:= time(): fminimax(S, 0.0001, 1 .. 8, 1 .. 8, 1 .. 8, 'h'), time() - t, eval(h);

[0.95332226\*10^(-28), [8, 8, 8], 0.2625893241, [1, 1, 1], 0.594, table([1 = 2, 2 = {}, 3 = {}])

> H:= (x, y) -> x\*y + sqrt(x + y): fminimax(H, 0.001, -Pi .. Pi, -Pi .. Pi, 'h'); plots[display](h[3]);

[-9.8596, [-3.14, 3.14], 12.36559282, [3.14, 3.14]]



ecsolve - equations solving with limitations on solutions

Call format of the procedure:

ecsolve(E::{algebraic, equation, set({equation, algebraic})}, v::set(name) {, x=range, y= range, ... })

Formal arguments of the procedure:

**E** – a single equation or system of equations  
**v** – a set of single or several names (*unknowns to solve for*) {**x**, **y**, ...}  
**x = a..b, y = c..d, ...** – (*optional*) ranges defining intervals for solutions relative to unknowns {**x**, **y**, ...}

Description of the procedure:

For solution of a single equation or system of equations *Maple* has procedure *solve*. However, in a series of cases there is a problem of obtaining of the solution satisfying the given conditions, namely: values of solution of a single equation or system of equations should be in the given intervals. The given problem is solved by the procedure *ecsolve*.

The first argument **E** of the procedure defines algebraic expression, single equation or a set of equations and/or algebraic expressions. The second argument **v** defines a set of single name or several names (*unknowns to solve for*), for example {**x**, **y**, ...}. At last, optional arguments since the 3rd argument define intervals for resultant values of unknowns – solution of equation or system of equations. These arguments are coded as ranges: for example **x = a .. b** supposes that the obtained value for unknown **x** of the solution should be in interval [**a**, **b**], i.e. **a <= x <= b**. Coding of the first actual argument **E** is similarly to the case of standard procedure *solve*, whereas the second actual argument **v** is coded as a set even for case of single unknown. In addition, in contrast to the standard procedure *solve*, for the procedure *ecsolve* the second argument is obligatory. In principle, the second argument could be defined also optional, however: (1) its compulsory coding is not difficult for the user, and (2) the compulsory coding allows to simplify algorithm of the procedure essentially.

The procedure call *ecsolve*(**E**, **v**) with two actual arguments is practically equivalent to the call *solve*(**E**, **v**) with the difference that obtaining of the more reliable solution is supported. The solutions for **v** are returned as a set of equations. The output from *ecsolve* in general is a sequence of solutions. When *ecsolve* is unable to find any solutions, the *NULL* value is returned with output of appropriate warning. The procedure *ecsolve* is based on the standard procedure *solve*, therefore it inherits all basic features of the second one.

Coding of actual arguments, since the 3rd argument (*sequence of equations in the form of unknown = a .. b*) allows to obtain the solutions satisfying the above requirements. The procedure handles basic especial and erroneous situations. The examples below illustrate the use of the procedure.

```

ecsolve := proc(E::{algebraic, equation, set({algebraic, equation})}, v::set(name))
local a, b, k, j, p, res, t, cond, psi;
global _SolutionsMayBeLost;
  assign(res = solve(E, v), psi = ((x, y) -> `if` (x <= rhs(y) and lhs(y) <= x, `true`, `false`));
  if res = NULL then assign(`_SolutionsMayBeLost` = `true`), assign(`res` = solve(E, v));
    if res = NULL then return unassign(`_SolutionsMayBeLost`), WARNING("equation (system of
      equations) has no solutions, or `solve` was unable to find the solutions")
    end if
  end if;
  if nargs = 2 then return evalf(res)
  elif map(whattype, {args[3 .. -1]}) = {`= `} and map(type, {seq(evalf(rhs(args[k])),
    k = 3 .. nargs)}, 'range'('numeric')) = {`true`} and {seq(member(lhs(args[k]), v),
    k = 3 .. nargs)} = {`true`}
  then assign(`res` = evalf([res]), t = {}, cond = table([seq(k = -`infinity` .. `infinity`, k = v)])
  else error "some actual arguments since the 3rd are invalid"
  end if;

```

```

if nops(v) = 1 then
  for k in res do a := evalf(rhs(op(k)));
    if type(a, 'numeric') and psi(a, evalf(rhs(args[3]))) then t := {a, op(t)}
    end if
  end do;
  t
else assign(a = evalf([args[3 .. -1]]), seq(assign('cond[lhs(k)] = rhs(k)', k = a);
  for k in res do
    add('if'(type(rhs(k[j]), 'numeric') and psi(rhs(k[j]), cond[lhs(k[j]])), 0, 1), j = 1 .. nops(v));
    if % = 0 then t := {k, op(t)} end if
  end do;
  'if'(nops(t) = 0, {}, op(t))
end if
end proc

```

Typical examples of the procedure use:

```

> ecsolve({x^2 + y - 6 - z=0, x + y^2 + 5=0, z + x + y}, {x, y, z});
{x = 2.12807437 + 1.01847024*I, y = 0.19025338 - 2.67661554*I, z = -2.31832774 + 1.65814530*I}
> ecsolve({x^2 + y - 6 - z=0, x + y^2 + 5=0, z + x + y}, {x, y, z}, y=2 .. 7, z=4 .. 8);
{}
> a, b:= 8, 16: ecsolve({x^2*y^2 - b, x^2 - y^2 - a}, {x, y}, x= 1 .. 5, y= 1 .. 3);
{x = 3.10754795, y = 1.28718851}
> ecsolve({x^2*y^2 - b, x^2 - y^2 - a}, {x, y});
{x = 3.10754795, y = 1.28718851}, {x = -3.10754795, y = 1.28718851}

```

**spfn** - a special composition of functions

Call format of the procedure:

**spfn**(F::function, L::list)

Formal arguments of the procedure:

**F** - function of **n** arguments

**L** - list of functions and/or lists of functions of one argument

Description of the procedure:

The procedure *spfn* provides the composition of functions on arguments of the given function. The procedure call **spfn**(F(x1,x2, ..., xn), L[h1(y1),h2(y2), ..., hn(yn)]) returns the result of composition of functions from list L on arguments of function F as follows:

$$F(h_1(x_1), h_2(x_2), \dots, h_n(x_n))$$

In addition, if some element *j* of list L is list of functions of one argument, for example, **hj**=[**hj1**(y1), **hj2**(y2), ..., **hjk**(yk)], then the returned result will be of the form:

$$F(h_1(x_1), h_2(x_2), \dots, h_j(h_j(h_j2( \dots (h_jk(x_j)) \dots )), \dots, h_n(x_n))$$

If the list L has **p** elements less than quantity of arguments of F, only the first **p** arguments of F will be subjected to the above composition of functions, whereas if the list L has **p** elements more than quantity of arguments of F, superfluous elements of L are ignored. Examples below well illustrate the principle of composition of functions provided by procedure *cpfn*.

```

spfn := proc(F::function, L::list)
local a, b, d, n, m, k, j, z;

```

```

assign67(a = [], m = nops(L), n = [type(F, 'arity'(1)), `n-arity`][2], z = NULL);
for k to n do a := [op(a), op(k, F)] end do;
if n < m then b := L[1 .. n]
elif m < n then b := [op(L), seq(8, k = 1 .. n - m)] else b := L end if;
op(0, F)((proc(n)
  local k, j;
  for j to n do
    if b[j] = 8 then z := z, a[j]
    elif type(b[j], 'function') and type(b[j], 'arity'(1)) then z := z, `(op(0, b[j]))(a[j])
    elif type(b[j], 'list'('function')) and type(b[j], 'list'('arity'(1))) then
      z := z, `@(seq(k, k = map2(op, 0, b[j]))) (a[j])
    else error "arity > 1 and/or non-function in the 2nd actual argument"
    end if
  end do
end proc)(n), unassign(`n-arity`)
end proc

```

Typical examples of the procedure use:

```

> A:= a(x): B:= b(y): C:= c(z): H:= h(t): G:= g(v): V:= v(s): F:= f(x1, x2, x3, x4, x5):
  {spfn(F, [A, V, G, B, C, H, G])}, {spfn(F, [A, V, [G, B, C], H, G])}, {spfn(F, [A, V, [G, B, C]])};
  {f(a(x1), v(x2), g(x3), b(x4), c(x5))}, {f(a(x1), v(x2), g(b(c(x3))), h(x4), g(x5))},
  {f(a(x1), v(x2), g(b(c(x3))), x4, x5)}
> A:= a(x): B:= b(y): C:= c(z): H:= h(t): G:= g(v): V:= v(s): spfn(F, [A, V, [G, B, C], H, G]);
  f(a(x1), v(x2), g(b(c(x3))), h(x4), g(x5))
> {spfn(F, [A, V, [G, B, C], H, G])}, {spfn(F, [A, V, [G, B, C]])};
  {f(a(x1), v(x2), g(b(c(x3))), h(x4), g(x5))}, {f(a(x1), v(x2), g(b(c(x3))), x4, x5)}
> Z:= z(y): spfn(Z, [[A, V, G, B, C], H, G]);
  z(a(v(g(b(c(y))))))
> Z1:= z1(y, t): spfn(Z1, [[A, V, G, B, C], H, G]);
  z1(a(v(g(b(c(y))))), h(t))

```

**uvlama** - row-by-row merging of lists, vectors, arrays and matrixes into array or matrix

Call format of the procedure:

**uvlama**(M {, A1 {, A2 {, A3 ... {, Ap} ... } })

Formal arguments of the procedure:

**M** - a symbol {'array', 'matrix'}

**A1, ..., Ap** - Maple objects of type {array, list, matrix, vector}

Description of the procedure:

At operation with objects of type {array, list, matrix, vector} in some cases the problem of their row-by-row merging with the purpose of formation of objects of type {array, matrix} arises. Such row-by-row merging is made by addition of the last row (*list*) of **A<sub>j</sub>**-object by all rows (*list*) of **A<sub>j+1</sub>**-object. In addition, if lengths of rows (*lists*) of united objects are various, the merging is made within of the rows truncated to the minimal length.

At absence of the united **A<sub>j</sub>**-objects, the procedure call returns the empty object of **M**-type; if the actual **M**-argument is distinct from symbol {array, matrix} the erroneous situation is caused.

Whereas presence among the united objects of an object which is distinct from {array, list, matrix, vector}, the procedure call outputs the corresponding message.

The successful call of procedure *uwlama*(M {, A1 {, A2 {, A3 ... {, Ap} ...})) returns result of merging of objects A1, ... Ap as object of M-type. The examples presented below well illustrate the told. The procedure is useful enough in problems of linear algebra.

```

uwlama := proc(V::symbol)
local a, b, c, d, h, k;
  assign(a = {}, b = [], h = ((x::list, n::posint) -> x[1 .. n]));
if not member(V, {'array', 'matrix'}) then
  error "1st argument must be symbol {array, matrix} but has received <%1>", V
elif nargs = 1 then eval(V([]))
else for c in [args[2 .. -1]] do
  if type(c, {'array', 'list', 'matrix', 'vector'}) and type(c, {'list', 'array(1)', 'array(2)'})
  then
    d := [op(2, eval(c))]; a := {op(a), `if` (type(c, 'list'), nops(c), rhs(d[-1]))}[1]; b := [op(b), c]
  else WARNING("argument <%1> has a type different from {array, list, matrix, vector}
    or has dimensionality > 2", c)
  end if
end do
end if;
if b = {} then eval(V([])) else
  V(map(h, [seq(`if` (type(k, {'list', 'vector'}), convert(k, 'list1'), op(convert(k, 'list1'))), k = b)], a))
end if
end proc

```

Typical examples of the procedure use:

```

> L:= [AVZ,AGN,VSV,Ar,Art,Kr]: A:= array(1..3, 1..4, [[a1,b1,c1,d1], [x1,y1,z1,h1], [42,47,73,96]]):
> V:= vector(4, [v,g,s,k]): M:= matrix([[a,b,c,d], [x,y,z,h], [63,58,38,9]]): V1:= vector([1,2,3,4,5,6]):
> uwlama('matrix', Z, A, V, V1, G, M, V1);

```

Warning, argument <Z> has a type different from {array, list, matrix, vector} or has dimensionality > 2

Warning, argument <G> has a type different from {array, list, matrix, vector} or has dimensionality > 2

<i>a</i> 1	<i>b</i> 1	<i>c</i> 1	<i>d</i> 1
<i>x</i> 1	<i>y</i> 1	<i>z</i> 1	<i>h</i> 1
42	47	73	96
<i>v</i>	<i>g</i>	<i>s</i>	<i>k</i>
1	2	3	4
<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
<i>x</i>	<i>y</i>	<i>z</i>	<i>h</i>
63	58	38	9
1	2	3	4

```

> uwlama(RANS_IAN, Z, A, V, V1, G, M, V1);

```

Error, (in *uwlama*) 1st argument must be symbol {array, matrix} but has received <RANS\_IAN>

> `uvlama('matrix');` `type(%,'matrix'), type(%,'array'), Empty(%);`  $\Rightarrow$  [] *true, true, true*

**askeleton** - breadboards of Maple objects of type {array, Array, matrix, Matrix, vector, Vector}

Call format of the procedure:

**askeleton(A {, h})**

Formal arguments of the procedure:

**A** - Maple object of type {array, Array, matrix, Matrix, vector, Vector}

**h** - (optional) list or ser of Maple expressions

Description of the procedure:

In a series of the problems dealing with objects of type {array, Array, matrix, Matrix, vector, Vector} the expediency of visualization of a breadboard of such objects which in the certain measure would characterize elements of such object and their distribution in it. The module *LinearAlgebra* includes the special the Structured Data Browser to view large 1- or 2-dimensional Arrays, Matrices, or Vectors, or to display cross-sectional views of large multi-dimensional Arrays. For similar objects especially of traditional Maple type {array, matrix, vector} the package has no similar means. Therefore, procedure suggested below in the certain measure solves this problem.

The procedure call **askeleton(A {, h})** supposes up to two actual arguments. The obligatory argument **A** represents Maple object of type {array, Array, matrix, Matrix, vector, Vector}, whose breadboard presents the data structure as graphic object of corresponding dimension {(1xn) matrix, (nmx) matrix, (nmxp) parallelepiped}. Dimensionality of the object **A** not should exceed 3; otherwise, the erroneous situation arises. Breadboarding of object is made as follows:

# for elements of the object **A** that have zero values, the appropriate cells of the breadboard are painted by white color;

# for elements of the object **A** that have nonzero values, the appropriate cells of the breadboard are painted by aquamarine color;

# for non-evaluated elements of the object **A**, the appropriate cells of the breadboard are painted by yellow color. The given situation is characteristic for objects of type {array, matrix, vector} only, because for similar objects of NAG type non-evaluated elements automatically receive zero values at their creation;

# if the procedure call **askeleton(A, h)** uses the second argument **h** of type {set, list}, the breadboard cells corresponding to elements of object **A** with values from **h** are painted by red color.

In addition, for 3-dimensional objects of type {array, Array} the output breadboard is level-by-level representation of source object whose layers are divided by an interval, allowing rather conveniently to look through states of its internal cells. The given procedure allows to represent effectively enough distributions of values of Maple objects of the above-mentioned types. The examples below rather well illustrate the told.

```
askeleton := proc(A::{Matrix, Vector, matrix, vector, Array, array})
local a, b, c, d, k, j, m, n, p;
  `if (type(A, 'Matrix'), assign(a = [1 .. op(1, eval(A))[1], 1 .. op(1, eval(M))[2]]),
  `if (type(A, 'Vector'), assign(a = [1 .. op(1, eval(A))]), assign(a = [op(2, eval(A))])));
if nargs = 1 then assign(d = {0}) elif type(args[2], {'list', 'set'}) then assign(d = {0, op(args[2])})
  else error "2nd argument <%1> is invalid", args[2] end if;
if nops(a) = 1 then n := rhs(a[1]); assign67(c = seq(cat(`, k), k = 1 .. n)), unassign(c);
  seq(assign67(cat(`, k) = plottools['rectangle']([k - 1, round(1/10*n)], [k, 0],
```

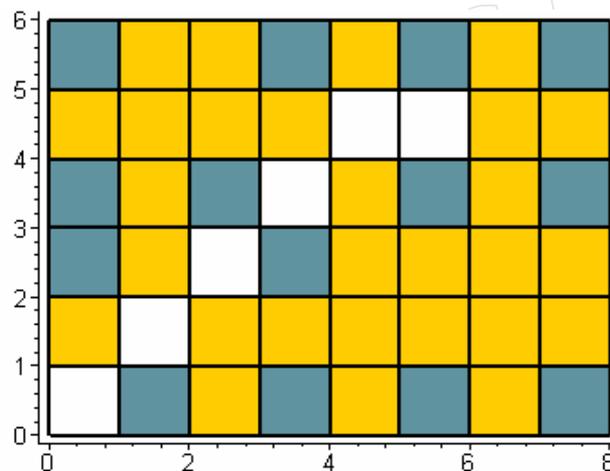
```

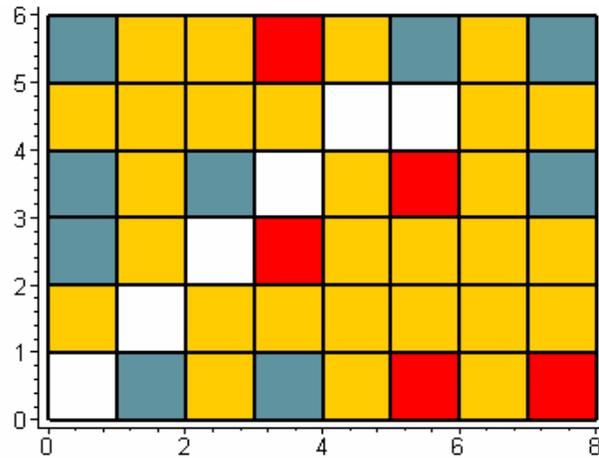
'color' = COLOR('RGB', `if` (A[k] = 0, op([1, 1, 1]), `if` (member(A[k], d), op([1, 0, 0]),
`if` (cat("", A, "[", k, "]") = convert(A[k], 'string'), op([1, 0.8, 0]), op([0.38, 0.58, 0.63]))))))),
k = 1 .. n);
plots[display](map(eval, {c}), 'thickness' = 2, 'scaling' = 'CONSTRAINED', 'axes' = 'FRAME'),
unassign(c)
elif nops(a) = 2 then n, m := rhs(a[1]), rhs(a[2]);
assign67(c = seq(seq(cat(`, k, "_", j), k = 1 .. n), j = 1 .. m)), unassign(c);
seq(seq(assign67(cat(`, k, "_", j) = plottools['rectangle']([j - 1, k], [j, k - 1],
'color' = COLOR('RGB', `if` (A[k, j] = 0, op([1, 1, 1]), `if` (member(A[k, j], d), op([1, 0, 0]),
`if` (cat("", A, "[", k, ",", j, "]") = convert(A[k, j], 'string'), op([1, 0.8, 0]),
op([0.38, 0.58, 0.63]))))))), k = 1 .. n), j = 1 .. m);
plots[display](map(eval, {c}), 'thickness' = 2, 'scaling' = 'CONSTRAINED', 'axes' = 'FRAME'),
unassign(c)
elif nops(a) = 3 then n, m, p := rhs(a[1]), rhs(a[2]), rhs(a[3]);
assign67(c = seq(seq(seq(cat(`, k, "_", j, "_", p), k = 1 .. n), j = 1 .. m), p = 1 .. p)), unassign(c);
seq(seq(seq(assign67(cat(`, k, "_", j, "_", p) = plottools['cuboid']([k, 2*j - 1, p],
[k + 1, 2*j, p + 1], 'thickness' = 2, 'color' = COLOR('RGB', `if` (A[k, j, p] = 0, op([1, 1, 1]),
`if` (member(A[k, j, p], d), op([1, 0, 0]), `if` (cat("", A, "[", k, ",", j, ",", p, "]") =
convert(A[k, j, p], 'string'), op([1, 0.8, 0]), op([0.38, 0.58, 0.63]))))))), k = 1 .. n),
j = 1 .. m), p = 1 .. p);
plots[display](map(eval, {c}), 'thickness' = 2, 'scaling' = 'CONSTRAINED', 'axes'='FRAME'),
unassign(c)
else error "array <%1> has dimensionality more than 3", A
end if
end proc
    
```

Typical examples of the procedure use:

```

> A:=array(1..6,1..8,[]): A[3,4]:=34: A[2,2]:=13: A[1,2]:=18: A[5,6]:=0: A[4,6]:=34: A[1,1]:=0: A[2,2]:=0:
A[3, 3]:=0: A[4, 4]:=0: A[5, 5]:=0: A[1, 6]:=34: A[6, 1]:=35: A[1,8]:=16: A[6,8]:=9: A[3,1]:=9: A[4, 8]:=9:
A[4,1]:=6: A[6, 4]:=16: A[1, 4]:=8: A[4, 3]:=7: A[6, 6]:=3: asceleton(A); asceleton(A, [16, 34]);
    
```





Procedure *askeleton* admits a series of modifications useful in a number of practical appendices.

**gpp** - generation of all simple univariate polynomials of the given degree  
**gpp1**

Call format of the procedures:

**gpp**(*x*::symbol, *n*::nonnegative {, *F*::{string, symbol}})

**gpp1**(*x*::symbol, *n*::nonnegative, *m*::nonnegative {, *F*::{string, symbol}})

Formal arguments of the procedures:

- x** - symbol defining name of leading variable of an univariate polynomial
- n** - a non-negative integer defining maximal degree of the polynomial
- m** - a non-negative integer defining residue ring **Zm**
- F** - (optional) name or full path to ASCII datafile for the generated polynomials

Description of the procedures:

The **gpp**(*x*, *n*) procedure generates all simple univariate polynomials in the variable *x* of degree *n* with coefficients from set {0, 1}. Such polynomials is widely used, for example, in signal processing, etc. The procedure call **gpp**(*x*, *n*) with two actual arguments returns table whose entries are the sought polynomials, whereas the procedure call **gpp**(*x*, *n*, *F*) with three actual arguments returns the NULL value, i.e. nothing; in addition, the sought polynomials are written into ASCII datafile defined by the third optional argument *F*.

The procedure **gpp1** extends the above procedure **gpp**, permitting the elements of residue ring **Zm** (equivalence classes), i.e. integers {0, 1, 3, ..., m-1} as coefficients of polynomials of degree *n*. The procedure call **gpp1**(*x*, *n*, *m*) with three actual arguments returns table whose entries are the sought polynomials of degree *n* with coefficients from **Zm**, whereas the procedure call **gpp1**(*x*, *n*, *m*, *F*) with four actual arguments returns the NULL value, i.e. nothing; in addition, the sought polynomials are written into ASCII datafile defined by the third optional argument *F*.

```

gpp := proc(x::symbol, n::nonnegative)
local a, b, c, d, k, j;
  assign(a = x^n, b = `if (2 < nargs and type(args[3], {'symbol', 'string'}), pathhf(args[3]), table([]));
  for k from 0 to 2^n - 1 do
    assign('c' = convert(convert(convert(k, 'binary'), 'string'), 'list1')),
    assign('d' = nops(c), 'c' = map(parse, c));
    if type(b, 'table') then b[k] := a + sort(sum(x^(d - j)*c[j], j = 1 .. d))
    else writeline(b, convert(a + sort(sum(x^(d - j)*c[j], j = 1 .. d)), 'string'))
    end if

```

```

end do;
if type(b, 'table') then eval(b) else close(b) end if
end proc
gpp1 := proc(x::symbol, n::nonnegative, m::nonnegative)
local a, b, c, d, k, j, invl;
  invl := proc(L::list)
    local k;
      [seq(L[nops(L) - k + 1], k = 1 .. nops(L))]
    end proc;
assign(b = `if` (3 < nargs and type(args[4], {'symbol', 'string'}), pathf(args[4]), table([])));
for k to m^(n + 1) do
  assign('c' = invl(convert(k, 'base', m))), assign('d' = nops(c));
  if d = n + 1 and c[1] <> 0 then
    if type(b, 'table') then b[k] := sort(sum(x^(d - j)*c[j], j = 1 .. d))
    else writeline(b, convert(sort(sum(x^(d - j)*c[j], j = 1 .. d)), 'string'))
    end if
  else next
  end if
end do;
if type(b, 'table') then eval(b) else close(b) end if
end proc

```

Typical examples of the procedures use:

```

> t:= time(): gpp(y, 15, "C:\\temp\\polynom"); time() - t;
24.328
> gpp1(x, 3, 2, "C:\\temp\\VSV"); gpp1(x, 3, 2);
table[8 = x3, 9 = x3 + 1, 10 = x3 + x, 11 = x3 + x + 1, 12 = x3 + x2, 13 = x3 + x2 + 1, 14 = x3 + x2 + x, 15 = x3 + x2 + x + 1]
> F:="C:\\temp\\VSV": fopen(F, READ): while not Fend(F) do parse(readline(F)) end do;
x3
x3+1
x3+x
x3+x+1
x3+x2
x3+x2+1
x3+x2+x
x3+x2+x+1

```

**lsf - data fitting by means of the least squares method**

Call format of the procedure:

**lsf(A, x, F, S::evaln)**

Formal arguments of the procedure:

- A** - (2xn) Array with numeric elements (*the fitted data*)
- x** - symbol (*name of leading variable of fitted algebraic expression*)
- F** - an algebraic expression from **x**-variable
- S** - an assignable name

Description of the procedure:

For data fitting by means of the method of least squares the procedure *lsf(A, y, x, F, S)* can be useful. It returns the equation defining resultant curve evaluated on the basis of source algebraic expression *F* and points of the source data given by a numerical *Array A* of dimensionality (2xn). The first and second row of the *A* Array define values of factor variable and resultant variable accordingly. Whereas via variable *S* the procedure returns the joint graph of found fitting curve and allocation of points of the source data *A*. In case of impossibility to calculate parameters of fitting expression *F* the procedure returns the *NULL* value with output of the appropriate message. The returned fitting expression does not support direct calculation of its values in the required points. For these purposes, for example, the construction *subs(x=b, expression)* can be used, where *b* - a required point. The procedure *lsf* has a series of useful appendices in data analysis.

```

lsf := proc(A::Array(numeric), x::symbol, F::algebraic, S::evaln)
local a, b, p, k, j, v, h, g, y;
  assign(a = rhs([ArrayDims(A)][2]), h = indets(F) minus {x});
  h := {seq('if (type(h[k], 'symbol'), h[k], NULL), k = 1 .. nops(h))};
  b := plots['pointplot']([seq([A[1, v], A[2, v]], v = 1 .. a)], 'symbol' = 'circle', 'symbolsize' = 14);
  g := fsolve({seq(diff(evalf(normal(simplify(expand(add((A[2, j] - subs(x = A[1, j], F))^2,
    j = 1 .. a))))), h[p]) = 0, p = 1 .. nops(h))}, h, 'fulldigits');
  try assign(y = ((x) -> eval(subs(g, F))), eval(subs(g, F)), assign(S = plots['display'](b, plot(y(x),
    x = A[1, 1] .. A[1, a]), 'thickness' = 2, 'axesfont' = ['TIMES', 'BOLD', 9]))
  catch "wrong number (or type) of parameters":
    WARNING("parameters %1 cannot be evaluated", h)
  end try
end proc

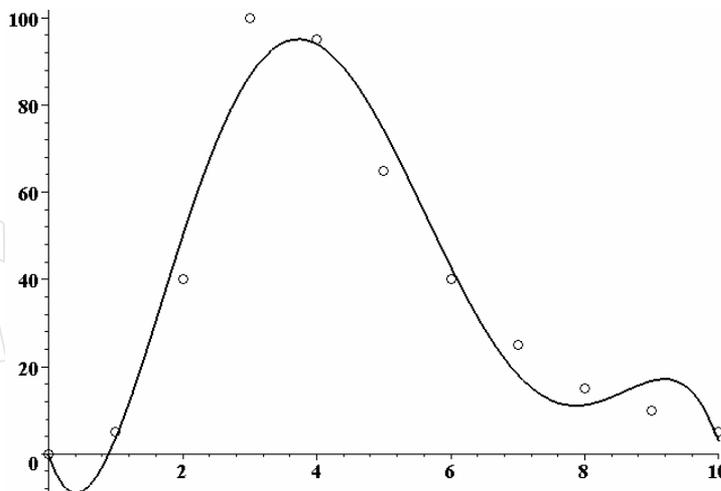
```

Typical examples of the procedure use:

```

> A:= Array(1 .. 2, 1 .. 11, [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [0, 5, 40, 100, 95, 65, 40, 25, 15, 10, 5]]):
  lsf(A, x, a*x^b*exp(-c*x), S);
  Warning, parameters {a, b, c} cannot be evaluated
> A:= Array(1 .. 2, 1 .. 11, [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [0, 5, 40, 100, 95, 65, 40, 25, 15, 10, 5]]):
  lsf(A, x, a*x^5 + b*x^4 + c*x^3 + d*x^2 + h*x + p, H); subs(x = 63, %); H;
-0.0817240592*x^5 + 2.16401955*x^4 - 19.6466462*x^3 + 66.3093028*x^2 - 44.8785384*x - 0.36810604
-51668458.24

```



**ListPack** – list of package modules in the Maple libraries

Call format of the procedure:

**ListPack**({F::mlib})

Formal arguments of the procedure:

**F** – (optional) a name or full path to a Maple library

Description of the procedure:

The *ListPack* procedure allows to obtain list of package modules contained in the main Maple library or a library structurally similar to the main library. The procedure *ListPack* handles all basic erratic and especial situations. The procedure call *ListPack*() supposes package modules will be sought in the main Maple library, whereas the procedure call *ListPack*(F) provides search in a Maple library whose name or full path is defined by actual argument F.

Generally, the successful procedure call returns two-element sequence of the sets, where the first set defines names of package modules of the first level whereas the second set defines modules of the second level which are submodules of modules of the first level. In addition, elements of the second set have one of the forms *name1:- name2*, *name1[name2]* and/or *name1/name2*, where *name1* and *name2* define names of modules of the first and second type accordingly. Unfortunately, at saving of the *mws*-document with *help*-page some of these designations are transformed. At absence of modules of the second level, the procedure call returns single set of modules names of the first level, whereas at absence of modules of the first level the procedure call returns the *NULL* value, i.e. nothing.

Along with a number of useful applications, the procedure *ListPack* allows to obtain more specific, in particular, the list of packages that are output on the command `? index, package`` for Maple 8. Procedure detects 5 modules of the first level, namely: *ExternalCalling*, *RootFinding*, *polytools*, *queue* and *priqueue* which are not reflected in the given list. Moreover, the procedure does not consider *Rif*, specified in the above list, as both the independent module, and a submodule of module *DEtools*.

```

ListPack := proc()
Local a, b, c, d, g, h, f, k, n, m, s, v, w, t, tst, gs, qt, p;
global _modpack;
  if 0 < nargs and type(args[1], 'mlib') then f := cat("", args[1], "/maple.hdb");
    if not belong(CF(args[1]), map(CF, [libname])) then
      assign67(p = 8, f = cat("", args[1], "/maple.hdb"), 'libname' = libname, args[1])
    end if;
    if type(f, 'file') then assign(d = {}, n = 120000, s = {}, v = {})
    else error "Maple library <%1> does not contain help database", args[1]
    end if
  else type(sin, 'libobj'), assign(d = {}, f = cat(_libobj, "/maple.hdb"), n = 120000, s = {}, v = {})
  end if;
  tst := proc(x) try parse(x); eval(%); parse(x) catch: NULL end try end proc;
  gs := proc(x::{set, list})
    local a, f;
    assign(f = cat([libname][1][1 .. 2], "/_$_Art16_Kr9$_"),
      assign(a = cat("_000:=proc(x::{list, set}) local a,b,c,k; global _modpack;
        assign(c = interface(warnlevel), '_modpack' = {}), interface(warnlevel = 0);
        for k in x do try b:= with(k); if type(b, 'list') then _modpack:= {op(_modpack), k}
        end if catch : NULL end try end do; _modpack,interface(warnlevel = c) end proc;

```

```

    _000(", convert(x, 'string'), "):"); writeline(f, a), close(f);
  read f;
  remove(f, _modpack, unassign('_000', '_modpack'),
    `if (p = 8, assign('libname' = op([libname][1 .. -2])), NULL)
  end proc;
  qt := (x) -> `if (x = {}, NULL, x);
  assign(m = filepos(f, `infinity`)), close(f), assign(b = open(f, 'READ'), h = {"with(", w = {});
  while filepos(b) < m do
    assign('a' = convert(subs({0 = NULL, 10 = NULL, 13 = NULL}, readbytes(b, n)), 'bytes')),
    assign('c' = Search2(a, h));
    if c <> [] then for k in c do
      g := Search2(a[k .. k + 35], {"", ":", " "});
      if g <> [] then d := {op(d), FNS(a[k + 5 .. k + g[1] - 2], " ", 3)} end if
    end do
  end if
end do;
seq(`if (search(k, ":", 't') or search(k, ":", 't'), assign('v' = {op(v), cat(`, k[1 .. t - 1]}),
  `if (search(k, "[", 't'), assign('v' = {op(v), cat(`, k[1 .. t - 1]}),
  'w' = {k, op(w)}), assign('v' = {op(v), cat(`, k)})), k = d);
assign('v' = {seq(`if (type(k, 'package'), k, NULL), k = v)}, 'w' = map(tst, w)), qt(v),
  `if (w = {}, NULL, qt(gs(w)))
end proc

```

Typical examples of the procedure use:

```

> ListPack("C:\\Program Files\\Maple 8\\Lib\\UserLib");
{plots, linalg, ExternalCalling, ListTools, SimpleStat, LinearAlgebra, DIRAX, plottools, SoLists, queue,
  ACC, Art_Kr, stats, priqueue, AlgLists, polytools, RootFinding, DEtools}

```

**marcmf** - an organization of archives of procedures and modules on the basis of *m*-files

Call format of the procedure:

**marcmf(F::{symbol, string}, P::{symbol, set(symbol), list(symbol)} {, R::symbol})**

Formal arguments of the procedure:

- F** - symbol or string defining a name or full path to a *m*-file
- P** - names of procedures and/or modules, or the order of means with the required date
- R** - (optional) work mode with elements of archive {'list', 'add', 'del', 'load'}

Description of the procedure:

Procedure *marcmf* supports work with archives of software on the basis of files of the internal *Maple* format (*m*-files). Such archives in a series of cases appear useful enough at work with sets of various versions of the same software created during various time. Procedure *marcmf* allows to create such archives, to update them, to delete their elements, to load into the current session the required means of archive and to receive contents of archives.

The first obligatory argument **F** defines a name of *m*-file with archive or full path to it. If the second argument **P** defines names of procedures and/or modules, at absence of the third optional argument **R** the procedure call *marcmf*(**F**, **P**) returns full path to the archive, creating or updating archive **F** by means defined in **P** argument. In addition, if **F** defines a nonexistent path, it is created with any level of nesting of subdirectories. Procedures are directly saved by the statement **save**

whereas before preservation of program modules their preliminary processing by procedures *mod21* and *modproc* is made. Whereas the procedure call *marcmf(F, P, 'add')* on the same first two arguments provides updating of **F** archive by means from **P** in the adding mode. Means are located in archive with indication of the current date in the form "**mm-dd-yyyy**".

The procedure call *marcmf(F, 'list')* returns the nested list of the following form:

$$[[N1, mm-dd-yyyy], [N2, mm-dd-yyyy], \dots, [Np, mm-dd-yyyy]]$$

where **Nj** - names of software and **dd-mm-yyyy** - dates of their saving. A single means is returned as the 2-element list. The call *marcmf(F, [Np, dd-mm-yyyy], 'load')* returns the name **Nj** of tool (with date of saving **dd-mm-yyyy**) of archive **F** which becomes accessible in the current session. In addition, the reference to procedures corresponds to the package agreements whereas before the first reference to exports of the program module saved in archive, it is necessary to execute the call **Nj()**. At absence of the required means in archive **F**, the procedure call initiates the appropriate erroneous situation.

The procedure call *marcmf(F, [Np, dd-mm-yyyy], 'del')* returns full path to a *m*-file - result of deletion out of the initial file **F** of means **Nj** (with date of saving **dd-mm-yyyy**) with output of the corresponding message. The resultant file is located in the same subdirectory, as the initial *m*-file, and its name is formed of name of the initial file with addition of prefix "\$". On other tuples of actual arguments the procedure call returns the *NULL* value, i.e. nothing. Procedure processes the basic special and erroneous situations, and admits in wide enough ranges the various kinds of updating (*optimization, extensions, etc.*). Examples of the fragment presented below well illustrate application of the procedure.

```
marcmf := proc(F::{symbol, string}, P::{symbol, set(symbol), list(symbol)})
local a, b, c, dt, k, f, t, p, h, rcs;
  assign67(c = {}, p = [], b = convert([1], 'bytes'), h = `|`);
  dt := proc()
    local a, b, c;
    assign(a = "_$artur16kr9$_"), assign(b = Mkdir(cat([libname][1][1 .. 2], "\\ ", a), 1),
      System(cat("Dir /T ", b, " > ", b));
    assign(c = readbytes(b, 'TEXT', `infinity`)), fremove(b);
    SLD(Red_n(c[nexts(c, a, "\n", 9)[2] + 1 .. nexts(c, a, "\n", 9)[1] - 1], " ", 2), " ")[1 .. 3];
    if nargs = 0 then %
      elif 0 < nargs and type(args[1], 'ssign') then
        cat(`if` (type(args[1], 'symbol'), ``, ""), seq(`if` (type(k, 'letter'), "",
          `if` (type(k, 'ssign'), args[1], k)), k = %[1]))
      end if
    end proc;
  rcs := proc(ss::{posint, symbol, string})
    local a, b, c;
    a := `if` (type(ss, 'posint') and belong(ss, 1 .. 31), convert([ss], 'bytes'),
      `if` (belong(convert(ss, 'bytes'), 1 .. 31), ss,
      ERROR("argument must be control symbol")));
    a := (proc() b := "_$Art16Kr9$_.m"; save a, b; readline(b); readline(b), fremove(b) end proc());
    null(search(a, "|", 'c')), a[c .. c + 1]
  end proc;
  seq(`if` (type(k, `module`), [mod21(k), modproc(k)], NULL), k = `if` (type(P, {'list', 'set'}), P, [P]));
  seq(assign('c' = {op(c), cat(k, b, dt("-"))}, cat(k, b, dt("-")) = eval(k)),
```

```

k = `if (type(P, {'list', 'set'}), P, [P]);
if not type(F, 'file') then
  if cat(" ", F)[-2 .. -1] = ".m" then f := F else f := cat(F, ".m") end if; Mkdir(f, 1);
  (proc) save args, f end proc(op(c)), unassign(op(c)), f
elif Ftype(F) = ".m" then a := Release1();
  if a <> SD(readbytes(F, 'TEXT', 2)[2]) then close(F);
    error "datafile <%1> is noncompatible with the current release", F
  else close(F)
  end if;
  if nargs = 2 then
    do a := readline(F);
      if a = 0 then close(F); break
      elif a[1] = "I" and search(a, h) then
        t := Iddn1(a); t := cat(`, a[searchtext(t, a) .. length(t) + 3]); p := [op(p), SLD(t, h)]
      end if
    end do;
    if args[2] = 'list' then p
    else [seq(`if (member(k[1], `if (type(P, {'list', 'set'}), P, [P])), k, NULL), k = p)]
    end if;
    if nops(%) = 1 then op(%) else % end if
  elif nargs = 3 and args[3] = 'add' and args[2] <> 'list'
  then f := cat([libname][1][1 .. 2], "\\_$_ArtKr$_.m"); h := cat(f, ".m");
    unassign(op(c)), procname(f, P);
    assign('a' = interface(warnlevel)), interface(warnlevel = 0), mmf(F, f, h),
    interface(warnlevel = a);
    null(writebytes(F, readbytes(h, 'infinity'))), close(F, h), remove(f, h), F
  elif nargs = 3 and args[3] = 'del' and type(args[2], 'list'('symbol')) and nops(args[2]) = 2
  then h := cat("", args[2][1], rcs(b), args[2][2]); h := cat("I", Iddn(length(h) - 1), h);
    assign('a' = readbytes(F, 'TEXT', `infinity`)), close(F);
    b := [op(Search2(a, {"\nI"})), 0];
    b := SUB_S([seq(`if (Suffix(a[b[k] + 1 .. b[k + 1] - 1], h, t),
      a[b[k] .. b[k + 1] - 1] = "", NULL), k = 1 .. nops(b) - 1)], a);
    assign('f' = CFF(F)), assign('f' = cat(seq(cat(k, "/"), k = f[1 .. -2]), "$", f[-1]));
    null(writebytes(f, `if (b[-1] = ",", b[1 .. -2], b))), close(f);
    WARNING("deletion result of <%1> is in datafile <%2>", args[2][1], f)
  elif nargs = 3 and args[3] = 'load' and type(args[2], 'list'('symbol')) and nops(args[2]) = 2
  then f := cat([libname][1][1 .. 2], "\\_$_ArtKr$_.m");
    h := cat("", args[2][1], rcs(b), args[2][2]); h := cat("I", Iddn(length(h) - 1), h);
    assign('a' = readbytes(F, 'TEXT', `infinity`)), close(F);
    b := [op(Search2(a, {"\nI"})), 0];
    b := [seq(`if (Suffix(a[b[k] + 1 .. b[k + 1] - 1], h, t), a[b[k] .. b[k + 1] - 1], NULL),
      k = 1 .. nops(b) - 1)];
    if b = [] then error "order %1 can't be executed", args[2] else b := b[1] end if;
    assign('f' = "_$Art16Kr9$_.m"), `if (b[-1] = ",", assign('b' = b[1 .. -2]), NULL);
    writebytes(f, cat(cat("M", Release1(), "R0"), b)), close(f); read f;
    remove(f), macro(args[2][1] = cat(args[2][1], convert([1], 'bytes'), args[2][2]))

```

```

end if
else error "datafile <%1> is not a m-file", F
end if
end proc

```

Typical examples of the procedure use:

```

> AVZ:=proc() `+(args)/nargs end proc: AGN:=module () export Res; Res:= () -> `+(args)/nargs
  end module: module VSV () export Res1; Res1:= () -> `+(args)/nargs end module:
> marcmf("C:\\Temp\\Academy\\Test.m", {AVZ, AGN, VSV});
  "C:\\Temp\\Academy\\Test.m"
> marcmf("C:\\Temp\\Academy\\Test.m", 'list');
  [[VSV, 11-24-2005], [AVZ, 11-24-2005], [AGN, 11-24-2005]]
> marcmf("C:\\Temp\\Academy\\Test.m", {AVZ, AGN}, 'add');
  "C:\\Temp\\Academy\\Test.m"
> marcmf("C:\\Temp\\Academy\\Test.m", 'list');
  [[VSV, 11-24-2005], [AVZ, 11-24-2005], [AGN, 11-24-2005], [AVZ, 11-24-2005], [AGN, 11-24-2005]]

```

**plotu** - create a two-dimensional plot of data with dimensions units on coordinates axes

Call format of the procedure:

```
plotu(Lx::list, Ly::list {, F::algebraic, x::symbol} {, popts})
```

Formal arguments of the procedure:

**Lx, Ly** - lists of data for axes **X** and **Y** accordingly  
**popts** - (*optional*) valid *plot* options  
**F, x** - (*optional*) algebraic expression and its leading variable accordingly

Description of the procedure:

In a series of cases it is necessary to plot curve point by point, determined by the dimensional data (*technical, physical data, etc.*). In addition, we should create a two-dimensional plot of data with the corresponding units on coordinates axes. For output of graphs of dependences the *Maple* has a number of means ('**plot**', '**plots**', etc.), however they deal with the dimensionless numeric data of *numeric-type*. Of course, they can be used and for the dimensional data, for example, the user can create graphs for the numerical data whereas their dimension units can be set through option '*labels*' of procedure '**plot**'. However, in a series of cases it causes the certain inconveniences. The following procedure **plotu** solves this problem in the certain extent.

Data for the procedure are coded in the form of lists of dimensional values; in addition, (1) numeric data should be of *realnum-type*, (2) a dimensional value is coded in the form **m\*Unit(<dimension unit>)**, where **m** - a *realnum-type* value, and (3) a data list should contain at least one dimensional value. Such approach enters unification to operation with means of module **Units** of the package. If any data list does not contain dimensional values, its data will belong to axes **X** or **Y** accordingly.

Optional actual arguments **popts** define *plot* options allowing to handle by appearance of output graphs. Within of these actual arguments the procedure call **plotu(Lx, Ly {, popts})** returns graphic representation of the dimensional data defined by lists **Lx, Ly** with dimension units on coordinates axes; in addition, standard procedure **plots[listplot]** is used.

Use of algebraic expression **F** with its leading variable **x** as actual arguments of the procedure call **plotu(Lx, Ly, F, x)** allows to smooth the data **Lx, Ly** by a curve **y=F(x)** by least-squares method. In both cases, the procedure call returns the sought data graph with dimension units on coordinates axes. The procedure **plotu** handles basic especial and erroneous situations. The procedure has many useful appendices at datafiles processing.

```

plotu := proc(Lx::list, Ly::list)
local a, b, c, Er, n, k, psi, _x, _y, y, z, v, X, Y, `type/realnum`, A, F, h, g, t, p;
  assign(n = min(op(map(nops, [Lx, Ly])), `type/realnum` = (N -> type(evalf(N), 'float'))));
  psi := proc(L::list, n::posint)
    local a, b, k;
    assign(a = [], b = {}, Er = ((a, b) -> ERROR("%1-th element of data list %2 is invalid", a, b)));
    for k to n do
      if type(L[k], 'numeric') then a := [op(a), L[k]]
      elif type(L[k], 'realnum') then a := [op(a), evalf(L[k])]
      elif whattype(L[k]) <> `` then
        if whattype(L[k]) = 'function' and op(0, L[k]) = 'Unit' then a := [op(a), 1];
          b := {op(b), convert(op(L[k]), 'symbol')}
        else Er(k, L)
        end if
      elif whattype(L[k]) = `` then
        if type(op(L[k])[1], 'numeric') then a := [op(a), op(L[k])[1]]
        elif type(op(L[k])[1], 'realnum') then a := [op(a), evalf(op(L[k])[1])]
        else Er(k, L)
        end if;
        if whattype(op(L[k])[2]) = 'function' and op(0, op(L[k])[2]) = 'Unit' then
          b := {op(b), convert(op(op(L[k])[2]), 'symbol')}
        else Er(k, L)
        end if
      else Er(k, L)
      end if
    end do;
    a, b
  end proc;
  a := [psi(Lx, n), psi(Ly, n)];
  for k in [2, 4] do
    if nops(a[k]) = 1 then assign(`if` (k = 2, _y, _x) = op(a[k]))
    elif 1 < nops(a[k]) then error "units for %1-axis are incorrectly defined"
    else `if` (k = 2, assign(`_y` = Y), assign(`_x` = X))
    end if
  end do;
  c := `if` (2 < nargs, seq(`if` (type(args[k], 'symbol'),
    assign(z = args[k]), `if` (type(args[k], 'algebraic'), assign(F = args[k]),
    `if` (member(lhs(args[k]), {'labels', 'labeldirections'}),
    NULL, args[k])), k = 3 .. nargs), NULL);
  if type(F, 'symbol') then plots['listplot']([seq([a[1][k], a[3][k]], k = 1 .. n)], 'labels' = [_x, _y],
  'labeldirections' = ['HORIZONTAL', 'VERTICAL'], c)
  else assign(A = Array([a[1], a[3]]), h = indets(F) minus {z});
  h := {seq(`if` (type(h[k], 'symbol'), h[k], NULL), k = 1 .. nops(h))};
  g := fsolve({seq(diff(evalf(normal(simplify(expand(add((A[2, j] - subs(z = A[1, j], F))^2,
  j = 1 .. n))))), h[p]) = 0, p = 1 .. nops(h)), h, 'fulldigits');
  try assign(y = ((x) -> eval(subs(g, F))), eval(subs(g, F))

```

```

catch "wrong number (or type) of parameters":
  WARNING("parameters %1 cannot be evaluated", h)
end try;
plot(y(x), x = A[1, 1] .. A[1, n], 'labels' = [_x, _y],
  'labeldirections' = ['HORIZONTAL', 'VERTICAL'], c)
end if
end proc

```

Typical examples of the procedure use:

```

> B:=[8*Unit(kg*m/s), 16*Unit(kg*m/s), 38*Unit(kg*m/s), 43*Unit(kg*m/s), 58*Unit(kg*m/s),
63*Unit(kg*m/s)]; A:=[1, 2*Unit(mass*length/time^2), 3, 4, 5, 6];

```

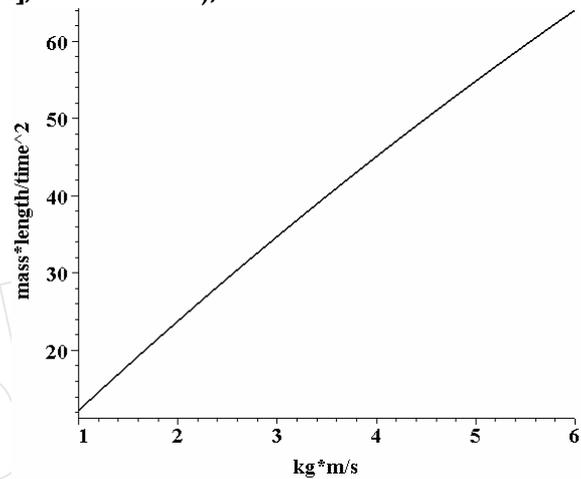
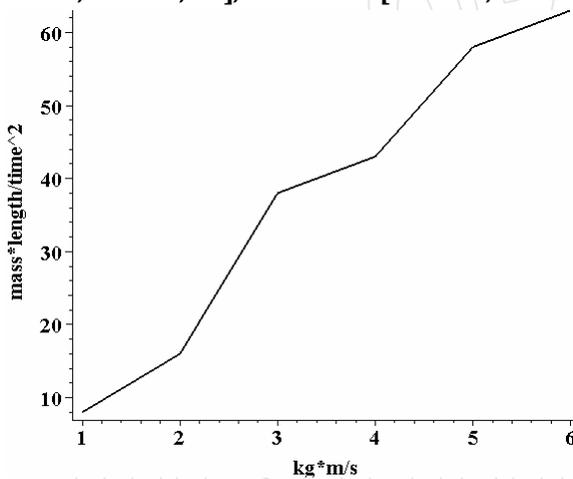
$$B := \left[ 8 \left[ \frac{\text{kg m}}{\text{s}} \right], 16 \left[ \frac{\text{kg m}}{\text{s}} \right], 38 \left[ \frac{\text{kg m}}{\text{s}} \right], 43 \left[ \frac{\text{kg m}}{\text{s}} \right], 58 \left[ \frac{\text{kg m}}{\text{s}} \right], 63 \left[ \frac{\text{kg m}}{\text{s}} \right] \right]$$

$$A := \left[ 1, 2 \left[ \frac{\text{mass length}}{\text{time}^2} \right], 3, 4, 5, 6 \right]$$

```

> plotu(A, B, labels=[a,b], color=green, labelfont=[TIMES, BOLD, 12], axesfont=[TIMES,
BOLD, 12], thickness=2); plotu(A, B, labels=[a, b], a*x*exp(b*x), x, color=red, labelfont=
[TIMES, BOLD, 12], axesfont=[TIMES, BOLD, 12], thickness=2);

```



### BootDrive - definition of the system drive of initial program loading

Call format of the procedure:

**BootDrive()**

Formal arguments of the procedure: *no*

Description of the procedure:

The procedure call **BootDrive()** returns the 2-element list whose first element defines a logic name of the drive of initial loading of operating system whereas the second element defines a full path to the main system directory. In addition, elements of the returned list are presented by values of *string*-type. The procedure handles basic especial and erroneous situations.

```

BootDrive := proc()
local a, b, c, v, k, t1, t2, f, D, x, y, w, G;
  D := (x, y) -> op({seq('if (search(x, k), RETURN(k), `false`), k = y)});
  G := proc(k, w)
    local n, h, z;
    z := cat(k, "\ \Boot.ini"); n := fopen(z, 'READ', 'TEXT');

```

```

while not Fend(n) do h := CF2(readline(n));
  if Search2(h, {"multi(" , "signature(") <> [] and search(h, cat(" " , w)) then return
    Close(z), map(Case, [k, cat(k, ":", FNS(sextr(h, "\\ \" , ssign)[1], " " , 3))], 'upper')
  end if
end do;
Close(z); error "system file BOOT.INI is corrupted"
end proc;
assign(a = map(CF2, {Adrive()}), t1 = {"95", "98", "me"}, t2 = {"2000", "2003", "nt", "xp"});
assign(f = cat([libname][1][1 .. 2], "/_ $Art16_Kr9$_"), system(cat("Ver > ", f));
assign(v = CF2(readbytes(f, 'TEXT', `infinity`))), delf1(f, `*`), assign(w = D(v, t1 union t2));
if member(w, t2) then
  for k in a do f := cat(k, ":\ PageFile.sys");
  try open(f, 'READ'); close(f)
  catch "file or directory does not exist": next
  catch "file I/O error": return G(k, w)
  catch "permission denied": return G(k, w)
  end try
end do
elif member(w, t1) then
  for k in a do f := cat(k, ":\ MsDos.sys");
  if not type(f, 'file') then next end if;
  do b := readline(f);
  if b = 0 then WARNING("file MSDOS.SYS on <%1> is corrupted", k); close(f); break
  end if;
  if search(CF2(b), "windir") then close(f); b := FNS(b[Search(b, "=")[1] + 1 .. -1], " " , 3);
  break
  end if
end do;
if b = 0 then next end if;
  try open(f, 'READ'); close(f)
  catch "file or directory does not exist": next
  catch "file I/O error": return map(Case, [b[1], b], 'upper')
  catch "permission denied": return map(Case, [b[1], b], 'upper')
  end try
end do;
error "fatal system error: correct file MSDOS.SYS has not been found on %1", map(Case, a, 'upper')
  else error "System error: unknown host operating system <%1>", v
  end if
end proc

```

Typical examples of the procedure use:

> BootDrive(); ⇒ ["C", "C:\WINDOWS"]

**mwsname** - name testing of the current Maple document

Call format of the procedure:

**mwsname()**

Formal arguments of the procedure: **no**

Description of the procedure:

In a series of cases, the problem of name testing of a current *Maple* document arises. The following procedure *mwsname* solves this problem. Successful procedure call *mwsname()* returns the name of the current *Maple* document as the symbol. The procedure implementation essentially uses the approach supported by two procedures *com\_exe1* and *com\_exe2*. Sole limitation is that the current *Windows* session should contains one *Maple* session only. As a rule, this condition is true. The procedure *mwsname* is rather useful at software processing of the current *Maple* documents.

```

mwsname := proc()
local a, b, c, t, k;
  unassign(VGS_vanaduspension_14062005);
  VGS_vanaduspension_14062005;
  assign('c' = "$Art16_Kr9$", 't' = interface(warnlevel)), assign('a' = system(cat("tlist.exe > ", c)));
  if a <> 0 then
    try interface(warnlevel = 0), com_exe2({tlist.exe})
    catch "programs appropriate": interface(warnlevel = t), RETURN(delf(c), false)
    end try;
    interface(warnlevel = t), goto(VGS_vanaduspension_14062005)
  end if;
  assign(b = fopen(c, 'READ'));
  while not Fend(c) do
    a := readline(c);
    if search(a, " Maple ") then k := Search2(a, {"[", "]});
      delf(c), assign('a' = a[k[1] .. k[-1]]);
      if search(a, "[Untitled", 'c') then RETURN(cat(`, a[c + 1 .. Search(a, ")"][1]))
      else
        c := Search(a, ".mws"); b := Search(a, "[");
        RETURN(cat(`, a[b[1] + 1 .. c[1] + 3]))
      end if
    end if
  end do
end proc

```

Typical example of the procedure use:

> **mwsname();** ⇒ *mwsname.mws*

The full set of software of the given orientation is represented in our books [239, 240] whereas the last release of library with these means is accessible to free-of-charge loading from website [259]. The second part of the offered book demonstrates the use of mathematical package *Maple* for solution of a whole series of important physical and engineering problems in such fields as thermal physics, heating engineering, elasticity theory, mechanics, hydrodynamics, hydromechanics, etc. The given problems were solved by applying the following numerical methods: finite element method and method of the characteristics in combination with functional means of mathematical package *Maple*, including means supported by our *Library* whose certain means have been presented above and in full measure can be found in our books [239, 240].

# Part 2.

## Application of *Maple* for solution of engineering-physical problems

*Finite Element Method (FEM)* was first developed in 1943 by R. Courant, who utilized the Ritz method of numerical analysis and minimization of variational calculus to obtain approximate solutions to vibration systems. Shortly thereafter, a paper published in 1956 by M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp established a broader definition of numerical analysis. By the early 70's, **FEM** was limited to expensive mainframe computers generally owned by the aeronautics, automotive, defense, and nuclear industries. Since the rapid decline in the cost of computers and the phenomenal increase in computing power, **FEM** has been developed to an incredible precision. Present day supercomputers are now able to produce accurate results for all kinds of parameters.

**FEM** consists of a computer model of a material or design that is stressed and analyzed for specific results. It is used in new product design, and existing product refinement. A company is able to verify a proposed design will be able to perform to the client's specifications prior to manufacturing or construction. Modifying an existing product or structure is utilized to qualify the product or structure for a new service condition. In case of structural failure, **FEM** may be used to help determine the design modifications to meet the new condition.

**FEM** uses a complex system of points called nodes which make a grid called a mesh. This mesh is programmed to contain the material and structural properties which define how the structure will react to certain loading conditions. Nodes are assigned at a certain density throughout the material depending on the anticipated stress levels of a particular area. Regions which will receive large amounts of stress usually have a higher node density than those which experience little or no stress. Points of interest may consist of: fracture point of previously tested material, fillets, corners, complex detail, and high stress areas. The mesh acts like a spider web in that from each node, there extends a mesh element to each of the adjacent nodes. This web of vectors is what carries the material properties to the object, creating many elements. A wide range of **FEM** applications, we uses the method for solving of numerous problems in mechanics, hydromechanics,

The *second* part of the offered book demonstrates the use of mathematical package *Maple* for solution of a whole series of important engineering-physical problems in such fields as: thermal physics, heating engineering, elasticity theory, mechanics, hydrodynamics, hydromechanics, etc. The given problems were solved by applying the following numerical methods: **FEM** and method of the characteristics in combination with functional facilities of mathematical package *Maple*.

# Chapter 8.

## The base problems of thermal conduction

The phenomenon of *thermal conduction* represents the process of distribution of thermal energy at immediate touch of separate particles of a body or separate bodies having various temperatures. Thermal conduction is being conditioned by moving microparticles of substance. In the gaseous medium energy is transferred by means of diffusion of molecules and atoms, and in liquids and solids-dielectrics – by means of elastic waves. While in metals, energy is mainly transferred by means of diffusion of free electrons. Generally speaking, any physical phenomenon is accompanied by change of the physical quantities essential to the given phenomenon, in space and in time.

The process of thermal conduction, as well as other kinds of heat exchange (*heat transfer, convection of a heat, heat radiation*), can take only place provided body temperature varies at various points. Generally, the process of heat transmission by thermal conduction in a solid is accompanied by change of temperature both in space and in time. Modern science and technology promote appearance of new substances with various physical properties, which depend on the temperature and other physical characteristics.

In a *homogeneous* medium which consists of pure substance or mixture of substances, the properties vary in space continuously. In the *multiphase* medium consisting of a number of single-phase parts, on boundaries of their division the properties vary by jumps. The heat is exchanged in the single-phase and multiphase mediums in different ways. In the *single-phase* medium, the physical properties at various points are identical at identical *temperature* and *pressure*, while in *nonuniform* mediums they vary. In an *isotropic* medium, the physical properties do not depend on the selected direction, while in an *anisotropic* medium some properties depend on the selected direction.

People distinguish *stationary* and *non-stationary* temperature fields. The problems of thermal conduction can be divided into *linear* and *nonlinear* ones. If thermo-physical parameters of the medium do not depend on temperature, the problem of thermal conduction is *linear*, otherwise – *nonlinear*. In the given chapter various problems of thermal conduction are considered in detail since their application is of utmost importance.

### 8.1. Linear stationary problem of thermal conduction

The *linear stationary problem* of thermal conduction is being solved when thermo-physical parameters of a substance do not depend on temperature and process of heat exchange is *steady*, i.e. temperature varies only in space of the substance. The *nonlinear problem* of thermal conduction at some point of temperature can also be ascribed to *linear* case. The *linear stationary problem* of thermal conduction is being solved together with problems of the theory of elasticity and plasticity.

#### 8.1.1. Calculated equations of the linear stationary process of heat exchange

We shall consider the *flat linear stationary* problem of thermal conduction. The heat conduction equation written in Cartesian or cylindrical coordinate systems, is of the following form:

$$\frac{k_x}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma \frac{\partial T}{\partial x} \right) + k_y \frac{\partial^2 T}{\partial y^2} = Q \quad (8.1)$$

where  $T$  – temperature;  $k_x, k_y$  – coefficients of thermal conductivity in the direction of axes  $X$  and  $Y$ ;  $Q$  – internal source of heat into unity of volume and time;  $\gamma=0$  at use of Cartesian coordinates and  $\gamma=1$  – cylindrical coordinates for the axially symmetric problem ( $x=r, y=z$ ). To solve the heat conduction equation, the *boundary conditions* defined in several ways should be known:

- Boundary conditions of the first kind. In this case the temperature on the surface of solid  $S_1$  (fig. 8.1) is allocated:

$$T = T_0(x, y) \quad (8.2)$$

- Boundary conditions of the second kind. In this case the values of heat flux  $q$  on some surface  $S_2$  and any moment of time (for a non-stationary problem) are set:

$$x^\gamma \left( k_x \frac{\partial T}{\partial x} l_x + k_y \frac{\partial T}{\partial y} l_y \right) = x^\gamma q \quad (8.3)$$

where  $l_x, l_y$  – directing cosines of the unit normal vector to the surface of a solid.

- Boundary conditions of the third kind. Temperature of a surrounding medium  $T_\infty$  and the law of heat exchange between body surface and surrounding medium are set. These boundary conditions characterize the law of heat exchange between a surface and surrounding medium during cooling and heating of the body. To describe the process of heat exchange between surface of body and surrounding medium the Newton-Rihman law is used [74]. The process of heat exchange between surface of body and surrounding medium is complex and depends on many parameters. According to the law of conservation of energy, the quantity of heat, which is output from unity of a surface into unit of time owing to a heat transfer, should be equal to the heat brought to the unit of surface into the unit of time owing to thermal conduction from interior body volume. Finally, the boundary condition of the third kind, given on surface  $S_3$  can be written as follows:

$$x^\gamma \left( \frac{\partial T}{\partial x} l_x + \frac{\partial T}{\partial y} l_y \right) = -x^\gamma h (T - T_\infty) \quad (8.4)$$

where  $h$  – heat transfer coefficient (the heat transfer coefficient  $h$  depends on the nature of the fluid, the geometry of the surface, and the dynamics of fluid motion past the surface);  $T_\infty$  – temperature of a surrounding medium.

- Boundary conditions of the fourth kind. They characterize conditions of heat exchange of a body with a surrounding medium under the law of thermal conduction. Bodies contact ideally (temperatures of the touching surfaces are identical). In this case equality of heat flux transits through a touching surface  $S_4$  (fig. 8.1):

$$x^\gamma \left( k_{x_1} \frac{\partial T_1}{\partial x} l_x + k_{y_1} \frac{\partial T_1}{\partial y} l_y \right) = x^\gamma \left( k_{x_2} \frac{\partial T_2}{\partial x} l_x + k_{y_2} \frac{\partial T_2}{\partial y} l_y \right) \text{ at } T_1 = T_2 \quad (8.5)$$

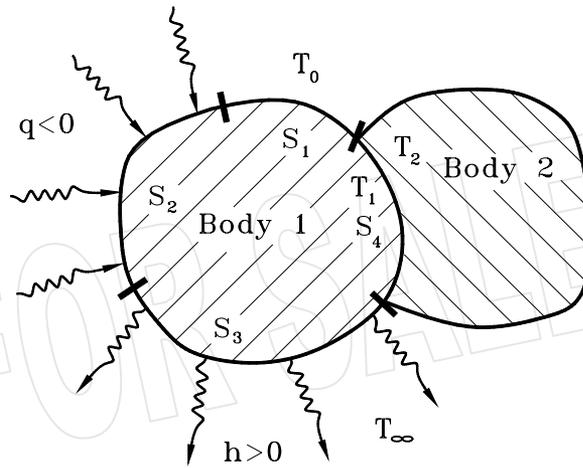


Fig. 8.1. The circuit of boundary conditions of the fourth order

Differential *heat conduction* equation (8.1) with boundary conditions (8.2) - (8.5) is solved by the *Finite Element Method (FEM)*, which is based on the idea of approximating the continuous function (in the given case - temperature) by a discrete model built on the set of piecewise continuous functions defined on a finite number of subdomains named as *elements* [75]. As function of the element, more often, the *multinomial* is applied. The order of a multinomial depends on the data used by a continuous function in each node of the element. To solve the considered problem, the *four-nodal finite element* is used (fig. 8.2).

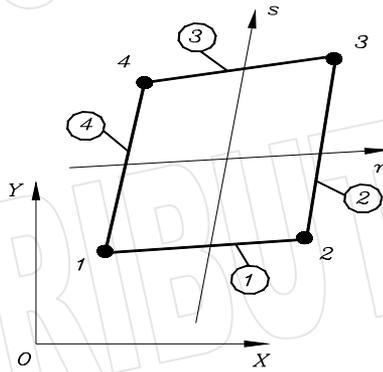


Fig. 8.2. A four-nodal isoparametric finite element (number in the circle indicates the number of the unit side)

Temperature  $T$  is approximated in all fields of a finite element by functions of the forms of the following view:

$$T(r, s) = \sum_{i=1}^4 N_i(r, s) T_i = [N(r, s)] \{T^{(e)}\}, \text{ where } N_i(r, s) - \text{shape function} \quad (8.6)$$

$$N_1(r, s) = \frac{1}{4}(1-r)(1-s); \quad N_2(r, s) = \frac{1}{4}(1+r)(1-s); \quad N_3(r, s) = \frac{1}{4}(1+r)(1+s); \quad N_4(r, s) = \frac{1}{4}(1-r)(1+s);$$

$$\{T^{(e)}\} = \begin{bmatrix} T_1^{(e)} & T_2^{(e)} & T_3^{(e)} & T_4^{(e)} \end{bmatrix}; \text{ where } \{T^{(e)}\} - \text{vector of nodal temperatures of a finite element} \quad (8.7)$$

The solution of differential *heat conduction* equation (8.1) with boundary conditions (8.2) - (8.5) can be changed by another problem, i.e. to find allocation of temperature  $T(x, y)$ , which minimizes the following functional:

$$J = \frac{1}{2} \int_V \left[ k_x \left( \frac{\partial T}{\partial x} \right)^2 + k_y \left( \frac{\partial T}{\partial y} \right)^2 - 2QT \right] x^\gamma dV + \int_{S_2} x^\gamma q T dA + \frac{1}{2} \int_{S_3} h x^\gamma (T - T_\infty)^2 dA, \quad (8.8)$$

*, where V - volume of explored field and A - area of the corresponding surface*

The whole field is divided into number NE of *isoparametric finite elements*, then functional (8.8) can be presented as follows:

$$J = \sum_{e=1}^{NE} J^{(e)}, \text{ where } J^{(e)} - \text{functional, written for a finite e-element} \quad (8.9)$$

The expression for functional  $J^{(e)}$  has the same form as functional (8.8). The condition of minimization of value of the functional (8.9) accepts the following form:

$$\frac{\partial J}{\partial T} = \sum_{e=1}^{NE} \frac{\partial J^{(e)}}{\partial \{T^{(e)}\}^T} = 0 \quad (8.10) \quad \text{or} \quad \frac{\partial J^{(e)}}{\partial \{T^{(e)}\}^T} = 0 \text{ for each finite element} \quad (8.11)$$

By substituting expression (8.6) into (8.11) and by executing differentiation, we gain the following form of the finite element of the linear equations system:

$$[k^{(e)}] \{T^{(e)}\} = \{p^{(e)}\}, \quad \text{where: } [k^{(e)}] = [k_1^{(e)}] + [k_2^{(e)}]; \quad (8.12)$$

$$[k_1^{(e)}] = \int_{V^{(e)}} x^\gamma \left( k_x \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + k_y \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dV; \quad [k_2^{(e)}] = \int_{S_3} x^\gamma h [N]^T [N] dA;$$

$$\{p^{(e)}\} = \{p_1^{(e)}\} - \{p_2^{(e)}\} + \{p_3^{(e)}\}; \quad \{p_1^{(e)}\} = \int_{V^{(e)}} x^\gamma [N]^T Q dV; \quad \{p_2^{(e)}(T)\} = \int_{S_3^{(e)}} x^\gamma [N]^T q dA; \quad (8.13)$$

$$\{p_3^{(e)}\} = \int_{S_3^{(e)}} x^\gamma h [N]^T T_\infty dA$$

Hence, derivatives of functions of forms over coordinates are defined as follows:

where  $[J]$  - Jacobi matrix of the next form:

$$\begin{Bmatrix} \frac{\partial N}{\partial x} \\ \frac{\partial N}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N}{\partial r} \\ \frac{\partial N}{\partial s} \end{Bmatrix}, \quad (i = 1, \dots, 4) \quad (8.14)$$

$$[J] = \begin{bmatrix} \sum_{i=1}^4 \frac{\partial N_i}{\partial r} X_i & \sum_{i=1}^4 \frac{\partial N_i}{\partial r} Y_i \\ \sum_{i=1}^4 \frac{\partial N_i}{\partial s} X_i & \sum_{i=1}^4 \frac{\partial N_i}{\partial s} Y_i \end{bmatrix}; \quad (8.15)$$

$X_i, Y_i$  - global coordinates of nodes of finite elements

The elementary volume  $dV$  is calculated as follows:

$$dV = \Delta(r, s)^{1-\gamma} dx dy = \Delta(r, s)^{1-\gamma} \det[J] dr ds, \text{ where: } \Delta(r, s) - \text{thickness of an element};$$

$$\Delta(r, s) = [N(r, s)]\{\Delta\} \text{ and } \{\Delta\}^T = [\Delta_1 \Delta_2 \Delta_3 \Delta_4] \quad (8.16)$$

The elementary surface area  $dA$  is calculated according to the formula:

$$dA = \Delta(r, s)^{1-\gamma} d\Gamma, \text{ where: } d\Gamma = \sqrt{\left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2} dr \text{ for the sides 1 and 3 of a finite element,}$$

$$\text{and } d\Gamma = \sqrt{\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2} ds \text{ for the sides 2 and 4 of a finite element} \quad (8.17)$$

In addition, the matrix  $[k^{(e)}]$  is named as a *matrix of thermal conduction*. The system of common equations is formed of systems of linear equations for finite elements as follows:

$$[K]\{T\} = \{P\}, \text{ where:} \quad (8.18)$$

$$[K] = \sum_{e=1}^{NE} [k^{(e)}] \text{ and } \{P\} = \sum_{e=1}^{NE} \{p^{(e)}\} \quad (8.19)$$

To solve the system of the linear algebraic equations (8.18), it is necessary to introduce boundary conditions of the first kind, while the boundary conditions of the second and third kind are already taken into account in functional (8.8).

### 8.1.2. Input data for the solution of the problem

To solve the *linear stationary problem of thermal conduction* on a certain plane, *Heat\_st\_linear* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable **F**, necessary *qualifier* of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of groups of finite elements (*ngroup*) and the number of known temperatures (*nbond*) are coded. Those finite elements which have identical *thermal conductivities*  $k_x$  and  $k_y$  are included into the corresponding group.

In the *second* line, the number of the sides of finite elements, on which the coefficients of heat transfer (*ngradh*) are given, the heat flux (*ngradq*) and the number of finite elements, in which the interior sources of heat (*nq*) are known, are coded.

In the *third* line, print code of intermediate results (*kprint*), type of a problem (*ngama*), code of the system solution of algebraic equations (*ksolve*), the number of iterations (*niter*) which is necessary to solve the equations system, precision of the solution (*toler*) and the temperature of a surrounding medium (*tapl*) are coded. If *kprint=0*, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* coordinates, *ngama=0*; otherwise - in *cylindrical* coordinates. The system of equations can be solved by two means: if parameter *ksolve=0*, Maple-language is used in solving ``solve`` function; otherwise, the equations system is solved by the method of *conjugate gradients*.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where: *nnode* – number of nodes of the finite element, i.e. *nnode*=4} are coded, and also the number of the group, to which this finite element {array **Mgroup**(*nelem*)} belongs. After arrays **Mtop** and **Mgroup** the elements of **Lbond**(*nbond*) array are coded line by line. The line number and an element of **Lbond** array is recorded into each line of file. The numbers of nodes, in which the values of temperature are known, are recorded into each line of **Lbond** array.

After array **Lbond** the elements of **Lgradh**(*ngradh*, 3) array are coded line by line. The line number and elements of **Lgradh** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of the given element, on which the coefficient of heat transfer **h** are known, are coded into each line of **Lgradh** array.

After array **Lgradh** the elements of **Lgradq**(*ngradq*, 3) array are coded line by line. The line number and elements of **Lgradq** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of this element, on which the heat flux **q** is known, are recorded into each line of **Lgradq** array.

After array **Lgradq** the elements of **Lq**(*nq*) array are coded line by line. The line number and the element of **Lq** array are recorded into each line of the file. The numbers of finite elements, in which the interior sources of heat **Q** are known, are recorded into each line of **Lq** array. Behind array **Lq**, the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime* = 2), are recorded line by line into the data file. **x**-coordinates are recorded into the first column of the **Coord** array, while **y**-coordinates of nodes of finite elements are recorded into the second column. The number of a node and also its (**x**,**y**)-coordinates are recorded into each line of the file. Behind **Coord** array, the elements of **Param**(*ngroup*, 2) array are coded line by line. The line number of **Param** array and values of thermal conductivities **k<sub>x</sub>** and **k<sub>y</sub>** are recorded into each line of the datafile. The line number of **Param** array should strictly correspond to the number of group of finite elements.

Behind array **Param**, the elements of **Storis**(*nelem*, 4) array are recorded line by line. The line number of **Storis** array and values of thickness of a body in each node of a finite element are recorded into each line of the datafile. If the axial-symmetric problem of a *thermal conduction* (*ngama* = 1) is considered, the thickness of the elements is not taken into account and the given array is not coded. Behind array **Storis**, the elements of **Bond**(*nbond*) array are recorded line by line. The line number of **Bond** array and the value of temperature in a node are recorded into each line of the datafile. The lines of **Bond** array should strictly correspond to the lines of **Lbond** array. After array **Bond**, the elements of **Gradh**(*ngradh*, 2) arrays are recorded line by line. The line number of array **Gradh** and values of coefficients of a convective heat exchange **h** for two nodes which are placed on a side of a finite element, where there is a heat exchange, are recorded into each line. The lines of array **Gradh** should strictly correspond to the lines of array **Lgradh**.

After array **Gradh**, the elements of **Gradq**(*ngradq*, 2) array are recorded line by line. The line number of **Gradq** array and values of a heat flux **q** for two nodes which are placed on the side of a finite element, where there is a heat exchange, are recorded into each line. The lines of **Gradq** array should correspond to the lines of **Lgradq** array. After array **Gradq**, the elements of **Q**(*nq*) array are recorded line by line. The line number of **Q** array and the value of an interior source of heat which is located in a finite element, are recorded into each line. The lines of **Q** array should strictly correspond to the lines of **Lq** array.

If boundary conditions are lacking, one element (*for example*, 1) is recorded only into array **Lbond**, while value 0 is ascribed to element **Bond**[1] of the array. In case of lack of interior sources of heat

into  $Lq$  array only one element (for example, 1) is recorded, while 0-value is ascribed to element  $Q[1]$  of the array. In the datafile between the recorded arrays the lines of the comments with the names of appropriate arrays are located. When reading the information from the datafile, these text lines are skipped.

The schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngroup, nbond*  
*ngradh, ngradq, nq, kprint, ngama, ksolve, niter, toler, tapl*  
 Text line \*  
 Arrays *Mtop(nelem, mmode), Mgroup(nelem)*  
 Text line \*  
 Array *Lbond(nbond)*  
 Text line \*  
 Array *Lgradh(ngradh, 3)*  
 Text line \*  
 Array *Lgradq(ngradq, 3)*  
 Text line \*  
 Array *Lq(nq)*  
 Text line \*  
 Array *Coord(npoin, ndime)*  
 Text line \*  
 Array *Param(ngroup, 2)*  
 Text line \*  
 Array *Storis(nelem, 4)*  
 Text line \*  
 Array *Bond(nbond)*  
 Text line \*  
 Array *Gradh(ngradh, 2)*  
 Text line \*  
 Array *Gradq(ngradq, 2)*  
 Text line \*

### 8.1.3. Brief description of Heat\_st\_linear program to solve the problem

The *Heat\_st\_linear* program was programmed on Maple-language; it consists of the basic program and 30 procedures. All procedures can be divided into three groups: procedures for data entry, for calculation and output of results. Memory size which is necessary to solve the concrete problem and the time of its solution depend on the used number of finite elements as well as the number of nodes. The program calculates values of temperature in the nodes of finite elements. Calculation results are output on the monitor and are recorded into a file; therefore, a *qualifier* of a target file must be ascribed to variable *file\_rez1* of the program. The values of temperature in the nodes of finite elements are recorded into file *file\_rez1*.

### 8.1.4. An example of use of Maple-program Heat\_st\_linear

*Allocation of temperature* in a rectangular body consisting of four layers is considered here (fig. 8.3); the physical properties of the given layers vary. On the right side of the body the heat flux  $q = 500 \frac{W}{m^2}$  is given, on the left side there is a heat exchange with known heat transfer coefficient

$h = 40 \frac{W}{m^2 \cdot K}$ . Each layer is an *isotropic* material. Thermal conductivity is defined as follows:

$$k_{Cu} = 385 \frac{W}{m \cdot K}; k_{Al} = 237 \frac{W}{m \cdot K}; k_{Asbestocement} = 0,58 \frac{W}{m \cdot K}; k_{Steel} = 60,5 \frac{W}{m \cdot K}.$$

*Input data for the test example:* Temperature of a surrounding medium  $t_{apl}=300$  K; thickness of body 0.02 m.; the number of finite elements  $n_{elem}=128$ , the number of nodes  $n_{poin}=153$ , the number of groups of finite elements  $n_{group}=4$ ,  $n_{bond}= 1$ ,  $n_{gradh}=4$ ,  $k_{print}=0$ ,  $n_q=1$ ,  $n_{gradq}=8$ ,  $n_{gamma}=0$ ,  $k_{solve}=0$ ,  $n_{iter}=50$ ,  $t_{oler}=10^{-6}$ .

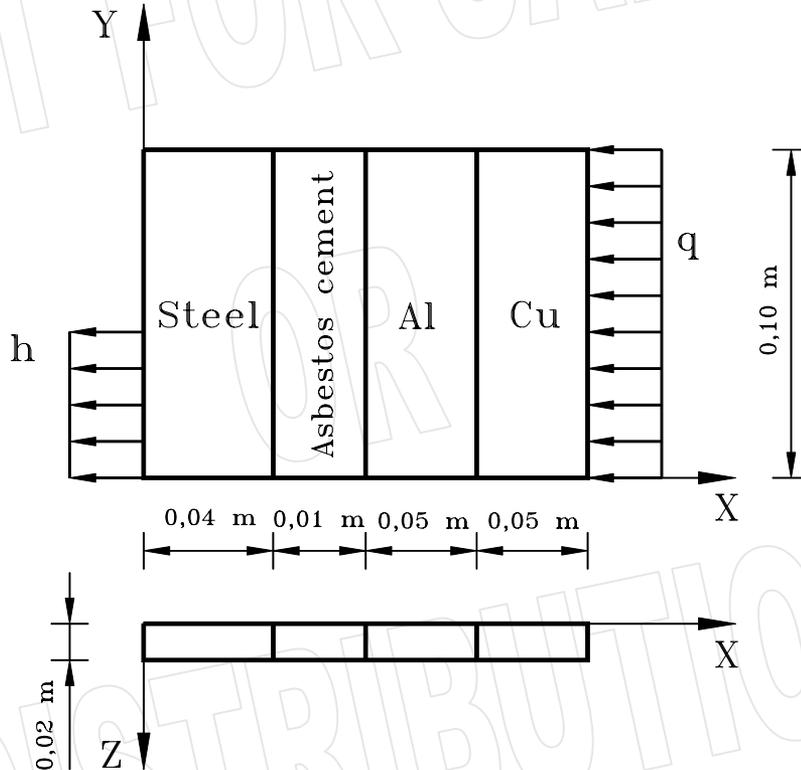


Fig. 8.3. The calculated scheme of an investigated body

**Results of calculation over the test example:**

Values of temperature (*Temp*) in nodes (*node*) of finite elements:

node=1	Temp=3.248099e+02	node=17	Temp=3.250912e+02
node=2	Temp=3.249402e+02	node=18	Temp=3.252038e+02
node=3	Temp=3.250354e+02	node=19	Temp=3.253031e+02
node=4	Temp=3.251000e+02	node=20	Temp=3.253765e+02
node=5	Temp=3.251667e+02	node=21	Temp=3.251009e+02
node=6	Temp=3.248026e+02	node=22	Temp=3.252005e+02
node=7	Temp=3.249373e+02	node=23	Temp=3.252718e+02
node=8	Temp=3.250496e+02	node=24	Temp=3.253419e+02
node=9	Temp=3.251357e+02	node=25	Temp=3.254733e+02
node=10	Temp=3.252220e+02	node=26	Temp=3.252922e+02
node=11	Temp=3.248522e+02	node=27	Temp=3.253016e+02
node=12	Temp=3.250023e+02	node=28	Temp=3.253626e+02
node=13	Temp=3.251223e+02	node=29	Temp=3.254365e+02
node=14	Temp=3.252327e+02	node=30	Temp=3.254949e+02
node=15	Temp=3.253148e+02	node=31	Temp=3.253466e+02
node=16	Temp=3.249592e+02	node=32	Temp=3.253772e+02

node=33	Temp=3.254295e+02	node=89	Temp=3.342022e+02
node=34	Temp=3.255068e+02	node=90	Temp=3.341289e+02
node=35	Temp=3.255758e+02	node=91	Temp=3.341563e+02
node=36	Temp=3.254031e+02	node=92	Temp=3.341817e+02
node=37	Temp=3.254205e+02	node=93	Temp=3.342063e+02
node=38	Temp=3.254754e+02	node=94	Temp=3.341277e+02
node=39	Temp=3.255523e+02	node=95	Temp=3.341573e+02
node=40	Temp=3.256342e+02	node=96	Temp=3.341865e+02
node=41	Temp=3.254479e+02	node=97	Temp=3.342092e+02
node=42	Temp=3.254586e+02	node=98	Temp=3.341278e+02
node=43	Temp=3.255140e+02	node=99	Temp=3.341548e+02
node=44	Temp=3.255976e+02	node=100	Temp=3.341786e+02
node=45	Temp=3.257612e+02	node=101	Temp=3.342071e+02
node=46	Temp=3.277222e+02	node=102	Temp=3.341281e+02
node=47	Temp=3.302412e+02	node=103	Temp=3.341506e+02
node=48	Temp=3.318775e+02	node=104	Temp=3.341700e+02
node=49	Temp=3.340981e+02	node=105	Temp=3.342017e+02
node=50	Temp=3.271190e+02	node=106	Temp=3.341245e+02
node=51	Temp=3.290344e+02	node=107	Temp=3.341465e+02
node=52	Temp=3.316722e+02	node=108	Temp=3.341730e+02
node=53	Temp=3.340959e+02	node=109	Temp=3.342008e+02
node=54	Temp=3.276757e+02	node=110	Temp=3.341288e+02
node=55	Temp=3.301311e+02	node=111	Temp=3.341483e+02
node=56	Temp=3.321480e+02	node=112	Temp=3.341772e+02
node=57	Temp=3.340981e+02	node=113	Temp=3.342064e+02
node=58	Temp=3.276857e+02	node=114	Temp=3.341266e+02
node=59	Temp=3.297173e+02	node=115	Temp=3.341546e+02
node=60	Temp=3.319555e+02	node=116	Temp=3.341854e+02
node=61	Temp=3.340957e+02	node=117	Temp=3.342124e+02
node=62	Temp=3.275669e+02	node=118	Temp=3.342182e+02
node=63	Temp=3.298936e+02	node=119	Temp=3.342311e+02
node=64	Temp=3.319903e+02	node=120	Temp=3.342508e+02
node=65	Temp=3.341000e+02	node=121	Temp=3.342678e+02
node=66	Temp=3.278077e+02	node=122	Temp=3.342209e+02
node=67	Temp=3.298144e+02	node=123	Temp=3.342369e+02
node=68	Temp=3.319696e+02	node=124	Temp=3.342514e+02
node=69	Temp=3.341057e+02	node=125	Temp=3.342684e+02
node=70	Temp=3.279038e+02	node=126	Temp=3.342235e+02
node=71	Temp=3.303033e+02	node=127	Temp=3.342388e+02
node=72	Temp=3.321698e+02	node=128	Temp=3.342515e+02
node=73	Temp=3.341037e+02	node=129	Temp=3.342668e+02
node=74	Temp=3.274182e+02	node=130	Temp=3.342251e+02
node=75	Temp=3.292493e+02	node=131	Temp=3.342380e+02
node=76	Temp=3.317393e+02	node=132	Temp=3.342519e+02
node=77	Temp=3.341008e+02	node=133	Temp=3.342678e+02
node=78	Temp=3.280097e+02	node=134	Temp=3.342213e+02
node=79	Temp=3.304324e+02	node=135	Temp=3.342367e+02
node=80	Temp=3.319570e+02	node=136	Temp=3.342508e+02
node=81	Temp=3.340986e+02	node=137	Temp=3.342675e+02
node=82	Temp=3.341253e+02	node=138	Temp=3.342174e+02
node=83	Temp=3.341517e+02	node=139	Temp=3.342341e+02
node=84	Temp=3.341716e+02	node=140	Temp=3.342522e+02
node=85	Temp=3.342042e+02	node=141	Temp=3.342622e+02
node=86	Temp=3.341241e+02	node=142	Temp=3.342175e+02
node=87	Temp=3.341543e+02	node=143	Temp=3.342359e+02
node=88	Temp=3.341803e+02	node=144	Temp=3.342535e+02

node=145	Temp=3.342693e+02	node=151	Temp=3.342337e+02
node=146	Temp=3.342207e+02	node=152	Temp=3.342519e+02
node=147	Temp=3.342365e+02	node=153	Temp=3.342647e+02
node=148	Temp=3.342531e+02		
node=149	Temp=3.342714e+02		
node=150	Temp=3.342235e+02		

The allocation of temperature in the considered body is represented in *fig. 8.4*.

Temperature  $T(x,y)$  Distribution:

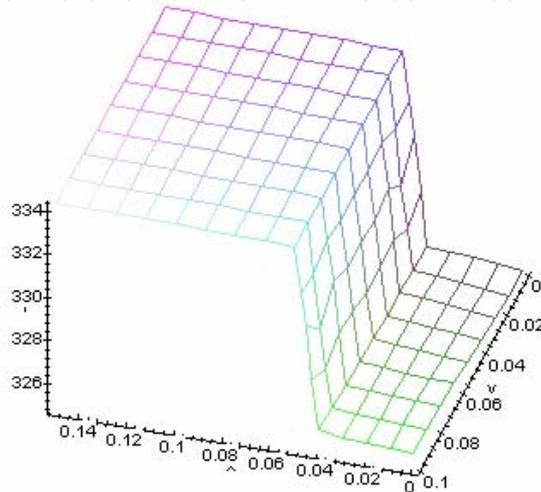


Fig. 8.4. Allocation of temperature in the investigated body

The *Heat\_st\_linear* program is intended for the solution of *stationary linear heat conduction equation* in an arbitrary flat body. The source code of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in PROBLEMS directory of archive attached to the book in files *Mb5\_1\_new.mws*, *Mb5\_1.dat* and *Mb5\_1\_1.rez* accordingly.

## 8.2. Nonlinear stationary problem of thermal conduction

The *nonlinear stationary problem of thermal conduction* is a more precise problem describing the process of heat exchange in the substance. The *nonlinearity* of the problem of thermal conduction is expressed; some thermo-physical parameters which are included in the heat conduction equation, and coefficients which are included in boundary conditions depend on *temperature*. The *thermal conductivities* of a whole series of substances depend on the temperature, and the heat transfer coefficients have rather composite temperature dependence on geometrical parameters of bodies, velocities of motion of adjoining media and a number of other parameters. When there is great difference in temperatures of a substance and surrounding medium, the heat exchange will considerably increase by means of *radiation*. The given thermo-physical process is characterized by the heat transfer coefficient of radiation, which is non-linearly temperature-dependent. The nonlinear stationary problem of thermal conduction can be decided together with other equations of mathematical physics.

### 8.2.1. Calculated equations of the nonlinear stationary process of heat exchange

When the dependence of parameters of thermo-physical properties of a material on temperature carries a *monotonic* character, functional approximation is used. The change of thermal conductivity of a material is usually well approximated by a *quadratic polynomial* of the following simple form:

$$\mathbf{k}(T) = \mathbf{k}_0 + \mathbf{k}_1 T + \mathbf{k}_2 T^2 \quad (8.20)$$

The dependence of coefficient of a surface convective heat exchange on temperature, in turn, is described by a polynomial of the fourth degree, namely:

$$h(T) = \sum_{i=0}^4 h_i T^i \quad (8.21)$$

Heat transfer coefficient of radiation has essentially nonlinear dependence on temperature, which is defined by the following relation:

$$h_r(T) = \chi \sigma (T^2 + T_\infty^2)(T + T_\infty) \quad (8.22)$$

where  $\chi$  - coefficient of *blackness*;  $\sigma$  - physical Stefan-Bol'tsman constant,  $\sigma = 5,6703 \cdot 10^{-8} \frac{W}{m^2 \cdot K^4}$ ;  $T_\infty$  - temperature of a surrounding medium.

The *nonlinear stationary problem* of thermal conduction is solved by FEM and is similar to the problem considered in the previous section. By fulfilling similar operations, in view of expressions (8.20 - 8.21), we obtain the system of *nonlinear algebraic expressions* written for a finite **e**-element:

$$[\mathbf{k}^{(e)}(T)]\{\mathbf{T}^{(e)}\} = \{\mathbf{p}^{(e)}(T)\}, \text{ where: } [\mathbf{k}^{(e)}(T)] = [\mathbf{k}_1^{(e)}(T)] + [\mathbf{k}_2^{(e)}(T)]; \quad (8.23)$$

$$[\mathbf{k}_1^{(e)}(T)] = \int_{V^{(e)}} x^\gamma \left( k_x(T) \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + k_y(T) \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dV;$$

$$[\mathbf{k}_2^{(e)}(T)] = \int_{S_3} x^\gamma h(T) [N]^T [N] dA; \quad \{\mathbf{p}^{(e)}(T)\} = \{\mathbf{p}_1^{(e)}(T)\} - \{\mathbf{p}_2^{(e)}(T)\} + \{\mathbf{p}_3^{(e)}(T)\}; \quad (8.24)$$

$$\{\mathbf{p}_1^{(e)}(T)\} = \int_{V^{(e)}} x^\gamma [N]^T Q dV; \quad \{\mathbf{p}_2^{(e)}(T)\} = \int_{S_3^{(e)}} x^\gamma [N]^T q dA; \quad \{\mathbf{p}_3^{(e)}(T)\} = \int_{S_3^{(e)}} x^\gamma h(T) [N]^T T_\infty dA$$

The common equations system is formed of systems of nonlinear equations of finite elements as follows:

$$[\mathbf{K}(T)]\{\mathbf{T}\} = \{\mathbf{P}(T)\} \quad (8.25)$$

The given system of nonlinear algebraic equations is solved by Newton method. With this purpose, by expanding vector

$$\{\mathbf{R}(T)\} = [\mathbf{K}(T)]\{\mathbf{T}\} - \{\mathbf{P}(T)\} \quad (8.26)$$

into a Taylor series in a point  $\{\mathbf{T}\}_i$ , and by keeping only the first two terms of the series, we obtain:

$$\{\mathbf{R}(T)\} \approx \{\mathbf{R}\}_i + \left[ \frac{\partial \mathbf{R}}{\partial T} \right]_i \{\Delta \mathbf{T}\}_i = \{\mathbf{0}\} \quad (8.27)$$

Then on each step of **i**-iteration, the system of linear equations of the following form is solved:

$$\left[ \frac{\partial \mathbf{R}}{\partial T} \right]_i \{\Delta \mathbf{T}\}_i = -\{\mathbf{R}\}_i \quad (8.28)$$

where  $\left[ \frac{\partial \mathbf{R}}{\partial \mathbf{T}} \right]$  – Jacobi matrix. Hence, the improved value of a vector of temperatures  $\{\mathbf{T}\}$  after  $i$ -iteration is defined by the following recursion relation:

$$\{\mathbf{T}\}_{i+1} = \{\mathbf{T}\}_i + \{\Delta \mathbf{T}\}_i \quad (8.29)$$

The iterative process is being continued up to realization of the following condition:

$$\max |\Delta T_j| \leq \varepsilon, \quad (j = 1, \dots, neq) \quad (8.30)$$

where  $\varepsilon$  – the given precision of the solution and  $neq$  – total number of the equations of the system.

### 8.2.2. Input data for the solution of the problem

To solve the *nonlinear stationary problem of thermal conduction* on a certain plane, *Heat\_st\_nonlinear* program is used. All data necessary for operation of this program, should be recorded into a file in advance; therefore, the necessary *qualifier* of the file is ascribed to variable **F**. The data in the file are placed in the strict order.

In the *first* line, the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of groups of finite elements (*ngroup*) and the number of known temperatures (*nbond*) are coded. Those finite elements which have identical *thermal conductivities*  $k_x$  and  $k_y$  are included into the corresponding group. In the *second* line, the number of the sides of finite elements, on which the coefficient of heat transfer (*ngradh*) are given, the coefficient of heat transfer of radiation (*ngradhr*), the heat flux (*ngradq*) and the number of finite elements, in which the interior sources of heat (*nq*) are known, are coded.

In the *third* line, a print code of intermediate results (*kprint*), a type of a problem (*ngama*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations' system, accuracy of the solution (*toler*) and the temperature of a surrounding medium (*tapl*) are coded. If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* coordinates, *ngama*=0; otherwise, in *cylindrical* coordinates. The equations' system can be solved in two ways: if parameter *ksolve*=0, the *solve* function of *Maple*-language is used; otherwise, the equations' system by the method of *conjugate gradients* is solved.

In the *fourth* line, the number of iterations (*niteration*) and accuracy (*tol*) of the solution of nonlinear equations are coded by Newton method. In each subsequent *nelem* line, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode*=4} are coded, and also the number of the group to which this finite element {array **Mgroup**(*nelem*)} belongs. After arrays **Mtop** and **Mgroup**, the elements of **Lbond**(*nbond*) array are coded line by line. The line number and an element of **Lbond** array is recorded into each line of the file. The numbers of nodes, in which the values of temperature are known, are recorded into each line of **Lbond** array.

After array **Lbond** the elements of **Lgradh**(*ngradh*, 3), array are coded line by line. The line number and elements of **Lgradh** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of the given element, on which the coefficient of heat transfer **h** are known, are coded into each line of **Lgradh** array.

After of array **Lgradh**, the elements of **Lgradhr**(*ngradhr*, 3) array are coded line by line. The line number and elements of **Lgradhr** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of this element, laying on a surface, where the

coefficients of a heat transfer of radiation are given, are recorded into each line of **Lgradhr** array.

After array **Lgradhr**, the elements of **Lgradq(ngradq, 3)** array are coded line by line. The line number and elements of **Lgradq** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of this element, on which the heat flux **q** is known, are recorded into each line of **Lgradq** array.

After array **Lgradq** the elements of **Lq(nq)** array are coded line by line. The line number and the element of **Lq** array are recorded into each line of the file. The numbers of finite elements in which the interior sources of heat **Q** are known are recorded into each line of **Lq** array into each line of **Lq** array. Behind array **Lq** into the data file the elements of **Coord(npoin, ndime)** array, where **ndime** - dimensionality of the problem (**ndime=2**), are recorded line by line. The **x**-coordinates and the **y**-coordinates of nodes of finite elements are recorded into the *first* and *second* columns of the **Coord** array. The number of a node and also its (**x,y**)-coordinates are recorded into each line of the file. Behind **Coord** array the elements of **Param(ngroup, 6)** array are coded line by line. The line number of **Param** array and six coefficients **k<sub>x0</sub>, k<sub>x1</sub>, k<sub>x2</sub>, k<sub>y0</sub>, k<sub>y1</sub>, k<sub>y2</sub>** are recorded into each line of the datafile, which approximate dependence of thermal conductivity in the direction of axes **X** and **Y**, accordingly, i.e.:

$$k_x(T) = \sum_{i=0}^2 k_{x_i} T^i \quad k_y(T) = \sum_{i=0}^2 k_{y_i} T^i$$

The line number of **Param** array should strictly correspond to the number of group of finite elements.

Behind array **Param** the elements of **Storis(nelem, 4)** array are recorded line by line. The line number of **Storis** array and values of thickness of a body in each node of a finite element are recorded into each line of the datafile. If the *axial-symmetric* problem of a *thermal conduction* (**ngama=1**) is considered, the thickness of elements is not taken into account and the given array is not coded. Behind array **Storis**, the elements of **Bond(nbond)** array are recorded line by line. The line number of **Bond** array and the value of temperature in a node are recorded into each line of the datafile. The lines of **Bond** array should strictly correspond to the lines of **Lbond** array. After the array **Bond** the elements of **Gradh(ngradh, 5)** arrays are recorded line by line. The line number of array **Gradh** and values of coefficients **h<sub>i</sub> (i=0..4)**, by which coefficient of a heat transfer is defined,

are recorded into each line, i.e.:  $h(T) = \sum_{i=0}^4 h_i T^i$ . The lines of array **Gradh** should strictly correspond to the lines of array **Lgradh**.

After array **Gradh** the elements of **Gradhr(ngradhr)** array are recorded line by line. The line number of **Gradhr** array and the value of coefficient of blackness **χ** of a finite element, are recorded into each line. The lines of **Gradhr** array should correspond to the lines of **Lgradhr** array. After array **Gradhr** the elements of **Gradq(ngradq, 2)** array are recorded line by line. The line number of **Gradq** array and values of a heat flux **q** for two nodes which are located on a side of a finite element, where heat is exchanged, are recorded into each line. The lines of **Gradq** array should strictly correspond to the lines of **Lgradq** array.

After array **Gradq** the elements of **Q(nq)** array are recorded line by line. The line number of **Q** array and the value of interior source of heat which is located in a finite element, are recorded into each line of the file. The lines of **Q** array should strictly correspond to the lines of **Lq** array.

If the boundary conditions are lacking, only one element (*for example, 1*) is recorded into array **Lbond** while value **0** is ascribed to element **Bond[1]** of the array. In case of lack of interior sources of heat into **Lq** array only one element (*for example, 1*) is recorded as well, while **0**-value is ascribed to element **Q[1]** of the array. In the datafile the lines of the comments with names of appropriate arrays are located between the recorded arrays. When reading the information from the datafile, these text lines are skipped.

*Schematic structure of the datafile:*

Text line \*  
*nelem, npoin, ngroup, nbond*  
*ngradh, ngradq, nq, kprint, ngama, ksolve, niteration, tol, tapl*  
Text line \*  
Arrays **Mtop(nelem, nnode), Mgroup(nelem)**  
Text line \*  
Array **Lbond(nbond)**  
Text line \*  
Array **Lgradh(ngradh, 3)**  
Text line \*  
Array **Lgradhr(ngradhr, 3)**  
Text line \*  
Array **Lgradq(ngradq, 3)**  
Text line \*  
Array **Lq(nq)**  
Text line \*  
Array **Coord(npoin, ndime)**  
Text line \*  
Array **Param(ngroup, 6)**  
Text line \*  
Array **Storis(nelem, 4)**  
Text line \*  
Array **Bond(nbond)**  
Text line \*  
Array **Gradh(ngradh, 5)**  
Text line \*  
Array **Gradhr(ngradhr)**  
Text line \*  
Array **Gradq(ngradq, 2)**  
Text line \*  
Array **Q(nq)**

### 8.2.3. Brief description of the Heat\_st\_nonlinear program solving the problem

The *Heat\_st\_nonlinear* program was programmed on *Maple*-language; it consists of the basic program and 41 procedures. All procedures can be divided into *three* groups: procedures for data entry, calculation and output of results. Memory size necessary for the solution of a concrete problem, and the time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of temperature in nodes of finite elements. Calculation results are output on the monitor and are recorded into a datafile; therefore a *qualifier* of a target file must be ascribed to variable *file\_rez1* of the program. The values of temperature in nodes of the finite elements are recorded into datafile *file\_rez1*.

### 8.2.4. An example of use of *Maple*-program Heat\_st\_nonlinear

The allocation of temperature in a rectangular body consisting of four various materials is considered as an important example: *copper (Cu)*, *aluminium (Al)*, insulating material - *asbestos-cement* and *steel* (fig. 8.5). On the other hand, heat flux  $q$  is known, from the left side a coefficient of a heat transfer  $h$  is given. From the upper part of explored body heat exchanges in the form of radiation  $h_r$ .

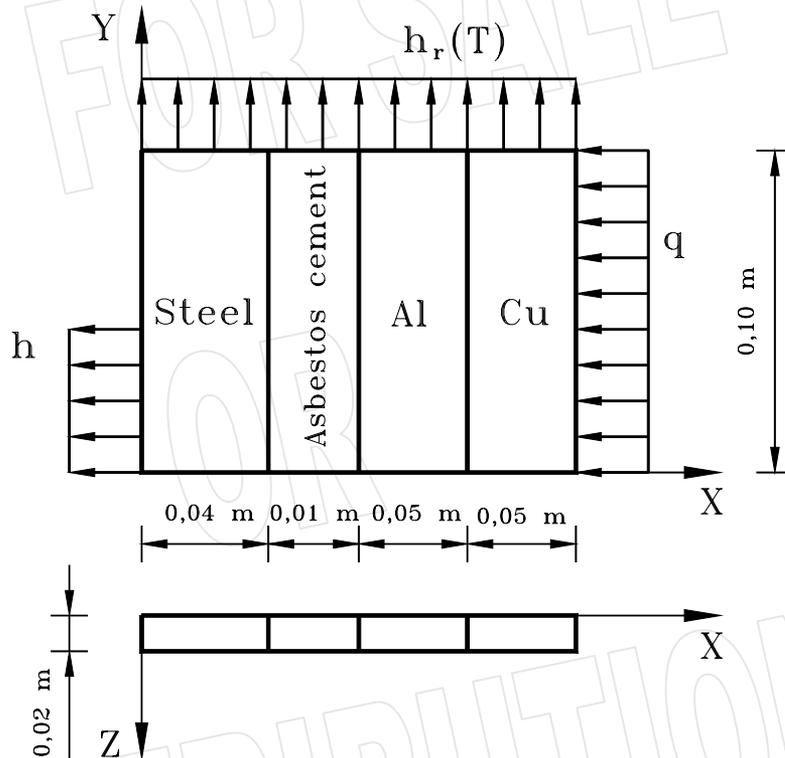


Fig. 8.5. The calculated scheme of the investigated body

The *thermal conductivities* of the used materials are calculated as follows:

$$\text{Cu: } k_x = k_y = 428,774 - 0,0916488 \cdot T + 1,44768 \cdot 10^{-5} \cdot T^2 \frac{\text{W}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 1000 \text{ K} ;$$

$$\text{Al: } k_x = k_y = 227,373 + 0,066194 \cdot T - 9,77612 \cdot 10^{-5} \cdot T^2 \frac{\text{W}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 800 \text{ K} ;$$

$$\text{Steel: (Mn} \leq 1\%, 0,1\% \leq \text{Si} \leq 0,6\%): k_x = k_y = 72,2536 - 0,0373551 \cdot T - 4,91872 \cdot 10^{-6} \cdot T^2 \frac{\text{W}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 1000 \text{ K} ;$$

$$\text{Asbestos-cement: } k_x = k_y = 0,58 \frac{\text{W}}{\text{m} \cdot \text{K}}$$

Input data for the test example: Temperature of surrounding medium  $t_{\text{apl}}=300 \text{ K}$ ; thickness of the body  $0.02 \text{ m}$ ; number of finite elements  $nelem=128$ , number of nodes  $npoin=153$ , number of groups of finite elements  $ngroup=4$ ,  $nbond=1$ ,  $ngradh=4$ ,  $kprint=0$ ,  $nq=1$ ,  $ngradq=8$ ,  $ngama=0$ ,  $ksolve=0$ ,  $niter=50$ ,  $toler=10^{-6}$ ,  $ngradhr=16$ ,  $ngradq=8$ ,  $niteration=50$ ,  $tol=10^{-6}$ . Heat flux  $q=100 \frac{\text{W}}{\text{m}^2}$  and

heat transfer coefficient  $h=40 \frac{\text{W}}{\text{m}^2 \cdot \text{K}}$ .

*Results of calculation over the test example:*

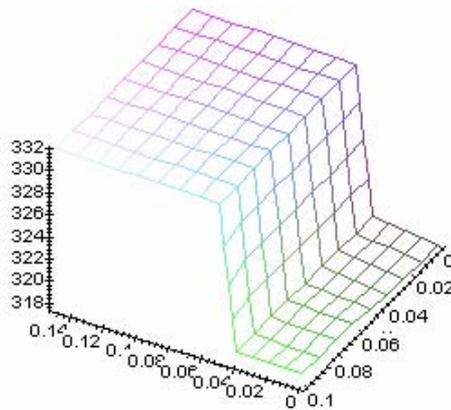
*Values of temperature (Temp) in nodes (node) of the finite elements:*

node=1	Temp=3.175062e+02	node=54	Temp=3.216214e+02
node=2	Temp=3.177227e+02	node=55	Temp=3.249548e+02
node=3	Temp=3.179125e+02	node=56	Temp=3.282876e+02
node=4	Temp=3.180767e+02	node=57	Temp=3.316202e+02
node=5	Temp=3.182164e+02	node=58	Temp=3.216803e+02
node=6	Temp=3.175277e+02	node=59	Temp=3.249929e+02
node=7	Temp=3.177440e+02	node=60	Temp=3.283053e+02
node=8	Temp=3.179326e+02	node=61	Temp=3.316174e+02
node=9	Temp=3.180955e+02	node=62	Temp=3.217493e+02
node=10	Temp=3.182346e+02	node=63	Temp=3.250372e+02
node=11	Temp=3.175951e+02	node=64	Temp=3.283248e+02
node=12	Temp=3.178091e+02	node=65	Temp=3.316132e+02
node=13	Temp=3.179932e+02	node=66	Temp=3.218168e+02
node=14	Temp=3.181506e+02	node=67	Temp=3.250805e+02
node=15	Temp=3.182872e+02	node=68	Temp=3.283453e+02
node=16	Temp=3.177152e+02	node=69	Temp=3.316074e+02
node=17	Temp=3.179286e+02	node=70	Temp=3.218551e+02
node=18	Temp=3.180921e+02	node=71	Temp=3.250966e+02
node=19	Temp=3.182358e+02	node=72	Temp=3.283406e+02
node=20	Temp=3.183673e+02	node=73	Temp=3.315996e+02
node=21	Temp=3.179774e+02	node=74	Temp=3.219580e+02
node=22	Temp=3.180957e+02	node=75	Temp=3.252176e+02
node=23	Temp=3.182156e+02	node=76	Temp=3.284426e+02
node=24	Temp=3.183376e+02	node=77	Temp=3.315890e+02
node=25	Temp=3.184618e+02	node=78	Temp=3.213255e+02
node=26	Temp=3.182384e+02	node=79	Temp=3.243468e+02
node=27	Temp=3.182600e+02	node=80	Temp=3.277254e+02
node=28	Temp=3.183358e+02	node=81	Temp=3.315736e+02
node=29	Temp=3.184358e+02	node=82	Temp=3.316636e+02
node=30	Temp=3.185526e+02	node=83	Temp=3.317060e+02
node=31	Temp=3.183511e+02	node=84	Temp=3.317495e+02
node=32	Temp=3.183706e+02	node=85	Temp=3.317942e+02
node=33	Temp=3.184246e+02	node=86	Temp=3.316630e+02
node=34	Temp=3.185101e+02	node=87	Temp=3.317054e+02
node=35	Temp=3.186217e+02	node=88	Temp=3.317489e+02
node=36	Temp=3.184038e+02	node=89	Temp=3.317936e+02
node=37	Temp=3.184200e+02	node=90	Temp=3.316613e+02
node=38	Temp=3.184682e+02	node=91	Temp=3.317036e+02
node=39	Temp=3.185457e+02	node=92	Temp=3.317472e+02
node=40	Temp=3.186535e+02	node=93	Temp=3.317919e+02
node=41	Temp=3.184034e+02	node=94	Temp=3.316583e+02
node=42	Temp=3.184191e+02	node=95	Temp=3.317007e+02
node=43	Temp=3.184653e+02	node=96	Temp=3.317443e+02
node=44	Temp=3.185401e+02	node=97	Temp=3.317891e+02
node=45	Temp=3.186349e+02	node=98	Temp=3.316540e+02
node=46	Temp=3.215692e+02	node=99	Temp=3.316963e+02
node=47	Temp=3.249208e+02	node=100	Temp=3.317401e+02
node=48	Temp=3.282717e+02	node=101	Temp=3.317852e+02
node=49	Temp=3.316223e+02	node=102	Temp=3.316481e+02
node=50	Temp=3.215826e+02	node=103	Temp=3.316905e+02
node=51	Temp=3.249296e+02	node=104	Temp=3.317347e+02
node=52	Temp=3.282758e+02	node=105	Temp=3.317802e+02
node=53	Temp=3.316218e+02	node=106	Temp=3.316402e+02

node=107	Temp=3.316831e+02	node=131	Temp=3.318455e+02
node=108	Temp=3.317278e+02	node=132	Temp=3.318753e+02
node=109	Temp=3.317740e+02	node=133	Temp=3.319061e+02
node=110	Temp=3.316302e+02	node=134	Temp=3.318130e+02
node=111	Temp=3.316739e+02	node=135	Temp=3.318419e+02
node=112	Temp=3.317194e+02	node=136	Temp=3.318717e+02
node=113	Temp=3.317667e+02	node=137	Temp=3.319025e+02
node=114	Temp=3.316190e+02	node=138	Temp=3.318082e+02
node=115	Temp=3.316629e+02	node=139	Temp=3.318372e+02
node=116	Temp=3.317087e+02	node=140	Temp=3.318672e+02
node=117	Temp=3.317583e+02	node=141	Temp=3.318980e+02
node=118	Temp=3.318217e+02	node=142	Temp=3.318024e+02
node=119	Temp=3.318503e+02	node=143	Temp=3.318317e+02
node=120	Temp=3.318800e+02	node=144	Temp=3.318618e+02
node=121	Temp=3.319107e+02	node=145	Temp=3.318927e+02
node=122	Temp=3.318211e+02	node=146	Temp=3.317957e+02
node=123	Temp=3.318498e+02	node=147	Temp=3.318254e+02
node=124	Temp=3.318795e+02	node=148	Temp=3.318557e+02
node=125	Temp=3.319102e+02	node=149	Temp=3.318866e+02
node=126	Temp=3.318195e+02	node=150	Temp=3.317887e+02
node=127	Temp=3.318482e+02	node=151	Temp=3.318186e+02
node=128	Temp=3.318779e+02	node=152	Temp=3.318489e+02
node=129	Temp=3.319086e+02	node=153	Temp=3.318798e+02
node=130	Temp=3.318168e+02		

On the *fig. 8.6* the allocation of temperature in the considered body is represented.

Temperature  $T(x,y)$  Distribution:



*Fig. 8.6.* Allocation of temperature in the investigated body

The *Heat\_st\_nonlinear* program is intended for the solution of a *stationary nonlinear heat conduction equation* in an arbitrary flat body. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb5\_2\_new.mws*, *Mb5\_2.dat* and *Mb5\_2\_1.rez* accordingly.

### 8.3. Linear non-stationary problem of thermal conduction

The *linear non-stationary problem of thermal conduction* is solved when the action of exterior medium onto some physical process or substance varies in time considerably. This problem is also relevant because the action of an interior source of heat onto substance changes considerably. The *linear non-stationary* heat conduction equation is one of the basic equations describing a convective heat exchange and mass transfer occurring in physical systems of various nature.

### 8.3.1. Calculated equations of a linear non-stationary process of heat exchange

Transportation of heat at the expense of thermal conduction is considered when the temperature of a system (*body*) varies not only from one point to another but also in the course of time. Such processes of thermal conduction are called *non-stationary processes*. They occur when various materials are heated (*cooled*). *Non-stationary thermal processes* are always linked with change of internal energy or *enthalpy* of substance. The heat conduction equation for non-stationary process is of the following form [74]:

$$\rho c \frac{dT}{dt} = \frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma k_x \frac{\partial T}{\partial x} \right) + k_y \frac{\partial^2 T}{\partial y^2} + Q \quad (8.31)$$

where  $\rho$  - density of material;  $c$  - specific heat;  $k_x, k_y$  - thermal conductivity coefficients;  $T$  - temperature;  $Q$  - interior source of heat into unit of time and volume;  $t$  - time;  $\gamma=0$  at use of Cartesian coordinates and  $\gamma=1$  - cylindrical coordinates for an axially symmetric problem ( $x=r, y=z$ ).

A considered body (*or process*) can move with some velocity  $\vec{U}$ . Then a *complete derivative* of temperature in time is defined by the following relation:

$$\frac{dT(x, y, t)}{dt} = \frac{\partial T}{\partial t} + u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y} \quad (8.32)$$

where  $u_x, u_y$  - components of a body velocity vector in a direction of axes  $X$  and  $Y$ . By substituting expression (8.32) by (8.31), we obtain the heat conduction equation which takes into account the fact of body motion, namely:

$$\rho c \left( \frac{\partial T}{\partial t} + u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y} \right) = \frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma k_x \frac{\partial T}{\partial x} \right) + k_y \frac{\partial^2 T}{\partial y^2} + Q \quad (8.33)$$

The velocity of body motion can be known or is defined from the solution of equations of motion. To solve equation (8.33), it is necessary to know initial conditions, namely:

$$t = 0, \quad T = T_0(x, y) \quad (8.34)$$

and also boundary conditions of the *second* and the *third* kind.

Heat conduction equation (8.33) is solved by FEM, therefore, the four-nodal isoparametric finite element (*see section 8.1*) is used. Temperature  $T$  in a finite  $e$ -element is approximated by the following relation:

$$T(r, s) = [N(r, s)]^T \{T^{(e)}(t)\} \quad (8.35)$$

The solution of non-stationary differential heat conduction equation (8.31) with boundary conditions (8.2 - 8.5) can be changed by another problem, i.e. by finding allocation of temperature  $T(x, y, t)$ , which minimizes the following functional:

$$J = \frac{1}{2} \int_V \left[ k_x \left( \frac{\partial T}{\partial x} \right)^2 + k_y \left( \frac{\partial T}{\partial y} \right)^2 - 2T \left( Q - \rho c \frac{dT}{dt} \right) \right] x^\gamma dV + \int_{S_2} x^\gamma q T dA + \frac{1}{2} \int_{S_3} h x^\gamma (T - T_\infty)^2 dA \quad (8.36)$$

From the condition of *minimal* value for functional (8.36), which was written for a finite element, we obtain a system of linear differential equations of the form:

$$[m^{(e)}]\{\dot{T}^{(e)}\} + [k^{(e)}]\{T\} = \{p^{(e)}\}, \quad (8.37)$$

$$\text{where: } [m^{(e)}] = \int_{V^{(e)}} \rho c x^\gamma ([N]^T [N]) dV; \quad \{\dot{T}^{(e)}\} = \left\{ \frac{\partial T^{(e)}}{\partial t} \right\}; \quad [k^{(e)}] = [k_1^{(e)}] + [k_2^{(e)}] + [k_3^{(e)}];$$

$$[k_3^{(e)}] = \int_{V^{(e)}} \rho c x^\gamma [N]^T \left( u_x \left[ \frac{\partial N}{\partial x} \right] + u_y \left[ \frac{\partial N}{\partial y} \right] \right) dV \quad (8.38)$$

The remaining matrixes and vectors are defined from expressions (8.13). The matrix of view  $[m^{(e)}]$  is named *matrix of a heat capacity*. The system of common equations is formed of systems of linear differential equations of finite elements, namely:

$$[M]\{\dot{T}\} + [K]\{T\} = \{P\} \quad (8.39)$$

By solving the system of differential partial equations (8.39) in view of initial conditions (*the boundary conditions are included into a variation problem*), we obtain the sought approximate allocation of temperature in a body in time. The system of equations (8.39) is changed by the corresponding system of finite-difference equations. To solve equation (8.39), the well known method of trapezoids is used:

$$\{T_{t+\tau}\} - \{T_t\} = \frac{\tau}{2} (\{\dot{T}_{t+\tau}\} - \{\dot{T}_t\}) \quad (8.40)$$

where  $\tau$  - integration step. From expression (8.40) we obtain the velocity of change of temperature into arbitrary moment time  $t + \tau$  as the following equation:

$$\{\dot{T}_{t+\tau}\} = \{\dot{T}_t\} + \frac{2}{\tau} (\{T_{t+\tau}\} - \{T_t\}) \quad (8.41)$$

Having substituted expression (8.41) by (8.39), we obtain the finite-difference equation for definition of sought temperature dependence, namely:

$$\left[ \frac{2}{\tau} M + K \right] \{T_{t+\tau}\} = \{P_{t+\tau}\} + [M] \left( \frac{2}{\tau} \{T_t\} - \{\dot{T}_t\} \right) \quad (8.42)$$

### 8.3.2. Input data for the solution of the problem

To solve the *linear nonstationary problem of thermal conduction* on a certain plane, *Heat\_nonst\_linear* program is used. All data necessary for operation of this program, should be recorded into a file in advance; therefore, variable **F** the necessary *qualifier* of the file is ascribed. The data in the file are placed in strict order. In the *first* line, the number of finite elements (*parameter nele*), the number of nodes (*npoint*), the number of groups of finite elements (*ngroup*) and the number of known temperatures (*nbond*) are coded. Those finite elements which have identical *thermal conductivities*  $k_x$  and  $k_y$  are included in the corresponding group. In the *second* line, the number of sides of finite elements, where the heat transfer coefficients (*ngradh*) are given, the heat flux (*ngradq*) and the number of finite elements, in which the interior sources of heat (*nq*) are known, are coded.

In the *third* line, a print code of intermediate results (*kprint*), type of a problem (*ngama*), the code of system solution of algebraic equations (*ksolve*), the number of iterations (*niter*) which is necessary to solve the equations system, a precision of the solution (*toler*) and the temperature of surrounding medium (*tapl*) are coded. If *kprint*=0, intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* coordinates, *ngama*=0; otherwise -

in *cylindrical* coordinates. The system of equations can be solved by two means: if parameter  $ksolve=0$ , the `solve` function of *Maple*-language is used for the solution of algebraic equations' system; otherwise, the system of equations is solved by the method of *conjugate gradients*.

In the *fourth* line, the number of integration steps (*ntime*), an integration step (*dtime*) and velocities of body motion in the direction of axes **X** and **Y** (*velyx*, *veluy*) are coded. In each subsequent *nelem* lines, the number of the finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode*=4} are coded and also the number of the group, to which this finite element {array **Mgroup**(*nelem*)} belongs.

After arrays **Mtop** and **Mgroup**, the elements of **Lbond**(*nbond*) array are coded line by line. The line number and an element of **Lbond** array is recorded into each line of the file. The numbers of nodes, in which the values of temperature are known, are recorded into each line of **Lbond** array. After array **Lbond** the elements of **Lgradh**(*ngradh*, 3) array are coded line by line. The line number and elements of **Lgradh** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of the given element, on which the coefficients of heat transfer **h** are known, are coded into each line of **Lgradh** array.

After array **Lgradh**, the elements of **Lgradq**(*ngradq*, 3) array are coded line by line. The line number and elements of **Lgradq** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of this element, on which the heat flux **q** is known, are recorded into each line of **Lgradq** array. After array **Lgradq** the elements of **Lq**(*nq*) array are coded line by line. The line number and the element of **Lq** array are recorded into each line of the file. The numbers of finite elements, in which the interior sources of heat **Q** are known, are recorded into each line of **Lq** array.

Behind array **Lq** the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line into the data file. The **x**-coordinates and the **y**-coordinates of nodes of finite elements are recorded into the *first* and *second* columns of the **Coord** array. The number of a node and its (**x**,**y**)-coordinates are recorded into each line of the file. Behind **Coord** array, the elements of **Param**(*ngroup*, 4) array are coded line by line. The line number of **Param** array, density, specific heat and the values of thermal conductivities of substance in the direction of axes **X** and **Y** (**ρ**, **c**, **k<sub>x</sub>**, **k<sub>y</sub>**) are recorded into each line of the datafile. The line number of **Param** array should strictly correspond to the number of the group of the used finite elements.

Behind array **Param**, the elements of **Storis**(*nelem*, 4) array are recorded line by line. The line number of **Storis** array and values of thickness of a body in each node of a finite element are recorded into each line of the datafile. If *axial-symmetric* problem of *thermal conduction* (*ngama*=1) is considered, the thickness of elements is not taken into account and the given array is not coded. Behind array **Storis**, the elements of **Bond**(*nbond*) array are recorded line by line. The line number of **Bond** array and the value of temperature in a node are recorded into each line of the datafile. The lines of **Bond** array should strictly correspond to the lines of **Lbond** array. After array **Bond**, the elements of **Gradh**(*ngradh*, 4) arrays are recorded line by line. The line number of array **Gradh** and values of **h<sub>i</sub>** (*i*=0..3) coefficients defining the *heat transfer coefficient*, i.e.  $\mathbf{h}(T) = [\mathbf{h}_0 + \mathbf{h}_1 \sin(\mathbf{h}_2 T)] e^{\mathbf{h}_3 T}$ , are recorded into each line. The lines of array **Gradh** should strictly correspond to the lines of array **Lgradh**.

After array **Gradh** the elements of **Gradq**(*ngradq*, 4) array are recorded line by line. The line number of **Gradq** array and values of **q<sub>i</sub>** (*i*=0..3) coefficients defining the heat flux, i.e.  $\mathbf{q}(T) = [\mathbf{q}_0 + \mathbf{q}_1 \sin(\mathbf{q}_2 T)] e^{\mathbf{q}_3 T}$ , are recorded into each line. The lines of **Gradq** array should correspond to the lines of **Lgradq** array. After array **Gradq**, the elements of **Q**(*nq*) array are recorded line by line. The line number of **Q** array and the value of an interior source of heat which

is located in a finite element, are recorded into each line. The lines of **Q** array should strictly correspond to the lines of **Lq** array.

If boundary conditions are lacking, one element (*for example, 1*) is recorded only into array **Lbond**, while value **0** is ascribed to element **Bond[1]** of the array. In case of lack of interior sources of heat into **Lq** array only one element (*for example, 1*) is recorded; while **0**-value is ascribed to element **Q[1]** of the array. In the datafile the lines of the comments with names of appropriate arrays are located between the recorded arrays. When reading information from the datafile, these text lines are skipped.

Schematic structure of the datafile:

Text line \*

*nelem, npoin, ngroup, nbond, ngradh, ngradq, nq, kprint, ngama, ksolve, niter, toler, tapl, ntime, dtime, velux, veluy*

Text line \*

Arrays *Mtop(nelem, nnode), Mgroup(nelem)*

Text line \*

Array *Lbond(nbond)*

Text line \*

Array *Lgradh(ngradh, 3)*

Text line \*

Array *Lgradq(ngradq, 3)*

Text line \*

Array *Lq(nq)*

Text line \*

Array *Coord(npoin, ndime)*

Text line \*

Array *Param(ngroup, 4)*

Text line \*

Array *Storis(nelem, 4)*

Text line \*

Array *Bond(nbond)*

Text line \*

Array *Gradh(ngradh, 4)*

Text line \*

Array *Gradq(ngradq, 4)*

Text line \*

Array *Q(nq)*

### 8.3.3. Brief description of Heat\_nonst\_linear program solving the problem

The *Heat\_nonst\_linear* program was programmed in Maple-language; it consists of the basic program and 38 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and the time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of temperature and velocity of its change in nodes of finite elements in the course of time. The calculation results are output on the monitor and are recorded into a file; therefore a *qualifier* of a target file must be ascribed to variable *file\_rez1* of the program. The values of temperature and velocity of its change in the nodes of finite elements are recorded into file *file\_rez1*.

### 8.3.4. An example of use of Maple-program Heat\_nonst\_linear

Allocation of temperature in a rectangular body consisting of two various materials is considered as an example: copper (Al) and aluminium (Cu) (fig. 8.7). Hence, the physical properties of layers composing a body vary.

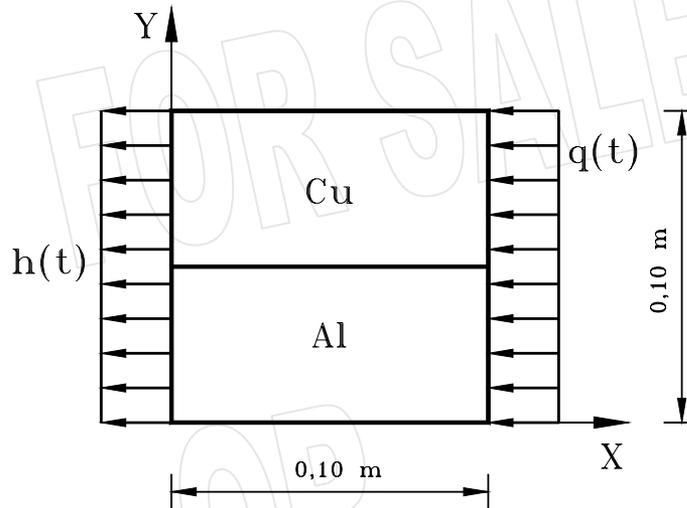


Fig. 8.7. The calculated scheme of the investigated body

It is supposed that the body moves in the direction of axis X with velocity 1 m/s. On the right side of the body, heat flux  $q(t)$  is given, on the left side heat is exchanged with the known heat transfer coefficient  $h(t)$ . Density, heat specific heat and thermal conductivities of materials at temperature  $T=300$  K are defined as follows:

$$\begin{array}{lll} \text{Cu: } \rho = 8933 \frac{\text{kg}}{\text{m}^3}; & c = 385 \frac{\text{J}}{\text{kg} \cdot \text{K}} & k_x = k_y = 401 \frac{\text{W}}{\text{m} \cdot \text{K}} \\ \text{Al: } \rho = 2702 \frac{\text{kg}}{\text{m}^3}; & c = 903 \frac{\text{J}}{\text{kg} \cdot \text{K}} & k_x = k_y = 237 \frac{\text{W}}{\text{m} \cdot \text{K}} \end{array}$$

The coefficients for definition of heat transfer and the heat flux are defined by the following relations, accordingly:

$$\begin{array}{llll} h_0 = 40 \frac{\text{W}}{\text{m}^2 \cdot \text{K}} & h_1 = 0 & h_2 = 0 & h_3 = 0 \\ q_0 = -5 \cdot 10^6 \frac{\text{W}}{\text{m}^2} & q_1 = 0 & q_2 = 0 & q_3 = -50 \end{array}$$

Input data for the test example: Temperature of surrounding medium  $tapl=300$  K; the number of finite elements  $nelem=72$ , the number of nodes  $npoint=90$ , the number of groups of finite elements  $ngroup = 2$ ,  $nbond=1$ ,  $ngradh=8$ ,  $kprint=0$ ,  $nq=1$ ,  $ngama=1$ ,  $niter=100$ ,  $ngradq=8$ ,  $ksolve=1$ ,  $niter=100$ ,  $toler=10^{-6}$ ,  $ntime=10^2$ ,  $dtime=10^{-3}$  s,  $velux=1$  m/s,  $veluy=0$ .

#### Results of calculation over the test example:

Values of temperature (Temp) and velocity of change of temperature (Dtempdt) in nodes (node) of finite elements (t=0.1 sec):

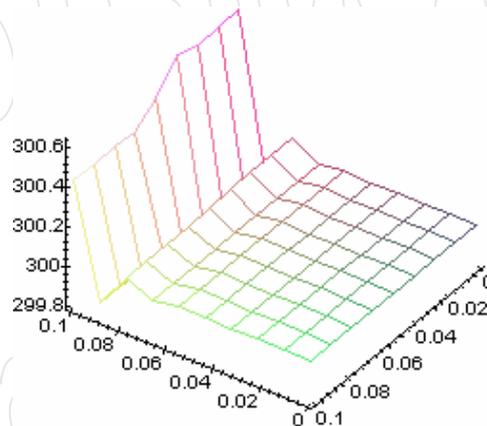
node=1	Temp=2.999998e+02	Dtempdt=-3.781853e-01
node=2	Temp=3.000001e+02	Dtempdt=3.172379e-01
node=3	Temp=2.999996e+02	Dtempdt=-6.485628e-01

node=4	Temp=3.000008e+02	Dtempdt=1.684846e+00
node=5	Temp=2.999976e+02	Dtempdt=-4.745696e+00
node=6	Temp=3.000069e+02	Dtempdt=1.390162e+01
node=7	Temp=2.999791e+02	Dtempdt=-4.169537e+01
node=8	Temp=3.000635e+02	Dtempdt=1.270460e+02
node=9	Temp=2.998042e+02	Dtempdt=-3.915353e+02
node=10	Temp=3.006085e+02	Dtempdt=1.217039e+03
node=11	Temp=2.999998e+02	Dtempdt=-3.801757e-01
node=12	Temp=3.000001e+02	Dtempdt=3.189071e-01
node=13	Temp=2.999996e+02	Dtempdt=-6.519711e-01
node=14	Temp=3.000008e+02	Dtempdt=1.693691e+00
node=15	Temp=2.999976e+02	Dtempdt=-4.770585e+00
node=16	Temp=3.000069e+02	Dtempdt=1.397446e+01
node=17	Temp=2.999790e+02	Dtempdt=-4.191363e+01
node=18	Temp=3.000638e+02	Dtempdt=1.277105e+02
node=19	Temp=2.998032e+02	Dtempdt=-3.935810e+02
node=20	Temp=3.006116e+02	Dtempdt=1.223392e+03
node=21	Temp=2.999998e+02	Dtempdt=-3.741583e-01
node=22	Temp=3.000001e+02	Dtempdt=3.138655e-01
node=23	Temp=2.999996e+02	Dtempdt=-6.416812e-01
node=24	Temp=3.000008e+02	Dtempdt=1.666999e+00
node=25	Temp=2.999976e+02	Dtempdt=-4.695503e+00
node=26	Temp=3.000068e+02	Dtempdt=1.375482e+01
node=27	Temp=2.999793e+02	Dtempdt=-4.125576e+01
node=28	Temp=3.000628e+02	Dtempdt=1.257086e+02
node=29	Temp=2.998062e+02	Dtempdt=-3.874201e+02
node=30	Temp=3.006021e+02	Dtempdt=1.204267e+03
node=31	Temp=2.999998e+02	Dtempdt=-3.963764e-01
node=32	Temp=3.000001e+02	Dtempdt=3.324668e-01
node=33	Temp=2.999996e+02	Dtempdt=-6.796303e-01
node=34	Temp=3.000008e+02	Dtempdt=1.765403e+00
node=35	Temp=2.999975e+02	Dtempdt=-4.972194e+00
node=36	Temp=3.000072e+02	Dtempdt=1.456392e+01
node=37	Temp=2.999781e+02	Dtempdt=-4.367830e+01
node=38	Temp=3.000665e+02	Dtempdt=1.330773e+02
node=39	Temp=2.997949e+02	Dtempdt=-4.100894e+02
node=40	Temp=3.006373e+02	Dtempdt=1.274609e+03
node=41	Temp=2.999998e+02	Dtempdt=-3.130169e-01
node=42	Temp=3.000001e+02	Dtempdt=2.627194e-01
node=43	Temp=2.999997e+02	Dtempdt=-5.373938e-01
node=44	Temp=3.000006e+02	Dtempdt=1.396711e+00
node=45	Temp=2.999980e+02	Dtempdt=-3.935901e+00
node=46	Temp=3.000057e+02	Dtempdt=1.153469e+01
node=47	Temp=2.999826e+02	Dtempdt=-3.461179e+01
node=48	Temp=3.000527e+02	Dtempdt=1.055098e+02
node=49	Temp=2.998373e+02	Dtempdt=-3.253103e+02
node=50	Temp=3.005058e+02	Dtempdt=1.011641e+03
node=51	Temp=2.999998e+02	Dtempdt=-2.543843e-01
node=52	Temp=3.000001e+02	Dtempdt=2.136124e-01
node=53	Temp=2.999997e+02	Dtempdt=-4.371543e-01
node=54	Temp=3.000005e+02	Dtempdt=1.136659e+00
node=55	Temp=2.999983e+02	Dtempdt=-3.204367e+00
node=56	Temp=3.000046e+02	Dtempdt=9.394573e+00
node=57	Temp=2.999858e+02	Dtempdt=-2.820118e+01
node=58	Temp=3.000430e+02	Dtempdt=8.600193e+01
node=59	Temp=2.998673e+02	Dtempdt=-2.652680e+02

node=60	Temp=3.004126e+02	Dtempdt=8.252496e+02
node=61	Temp=2.999998e+02	Dtempdt=-2.699928e-01
node=62	Temp=3.000001e+02	Dtempdt=2.266938e-01
node=63	Temp=2.999997e+02	Dtempdt=-4.638691e-01
node=64	Temp=3.000006e+02	Dtempdt=1.205996e+00
node=65	Temp=2.999983e+02	Dtempdt=-3.399501e+00
node=66	Temp=3.000049e+02	Dtempdt=9.965693e+00
node=67	Temp=2.999850e+02	Dtempdt=-2.991269e+01
node=68	Temp=3.000456e+02	Dtempdt=9.121244e+01
node=69	Temp=2.998593e+02	Dtempdt=-2.813121e+02
node=70	Temp=3.004375e+02	Dtempdt=8.750780e+02
node=71	Temp=2.999998e+02	Dtempdt=-2.657699e-01
node=72	Temp=3.000001e+02	Dtempdt=2.231526e-01
node=73	Temp=2.999997e+02	Dtempdt=-4.566337e-01
node=74	Temp=3.000005e+02	Dtempdt=1.187208e+00
node=75	Temp=2.999983e+02	Dtempdt=-3.346603e+00
node=76	Temp=3.000049e+02	Dtempdt=9.810799e+00
node=77	Temp=2.999852e+02	Dtempdt=-2.944829e+01
node=78	Temp=3.000448e+02	Dtempdt=8.979799e+01
node=79	Temp=2.998615e+02	Dtempdt=-2.769548e+02
node=80	Temp=3.004307e+02	Dtempdt=8.615392e+02
node=81	Temp=2.999998e+02	Dtempdt=-2.671653e-01
node=82	Temp=3.000001e+02	Dtempdt=2.243234e-01
node=83	Temp=2.999997e+02	Dtempdt=-4.590272e-01
node=84	Temp=3.000005e+02	Dtempdt=1.193426e+00
node=85	Temp=2.999983e+02	Dtempdt=-3.364119e+00
node=86	Temp=3.000049e+02	Dtempdt=9.862114e+00
node=87	Temp=2.999851e+02	Dtempdt=-2.960222e+01
node=88	Temp=3.000451e+02	Dtempdt=9.026707e+01
node=89	Temp=2.998607e+02	Dtempdt=-2.784006e+02
node=90	Temp=3.004330e+02	Dtempdt=8.660336e+02

Allocation of temperature in the considered body is represented on the *fig. 8.8*.

*Temperature T(x,y) Distribution:*



*Fig. 8.8. The allocation of temperature in the considered body*

The *Heat\_nonst\_linear* program is intended for the solution of a *non-stationary linear heat conduction equation* in an arbitrary flat body. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the *PROBLEMS* directory of archive attached to the book in files *Mb5\_3\_new.mws*, *Mb5\_3.dat* and *Mb5\_3\_1.rez* accordingly.

## 8.4. Nonlinear non-stationary problem of thermal conduction

The *nonlinear non-stationary problem of thermal conduction* is characterized by one of the basic equations of mathematical physics. This problem has both major self-maintained value for the definition of non-stationary temperature fields in substances as well as the complex problem of more composite physical problems, such as convective heat exchange and mass transfer and a whole series of other important problems.

### 8.4.1. Calculated equations of nonlinear non-stationary process of heat exchange

The *non-stationary problem of thermal conduction* is considered at nonlinear dependencies of thermo-physical parameters of substance from temperature (*specific heat*  $\mathbf{c}(\mathbf{T})$ , *thermal conductivity*  $\mathbf{k}(\mathbf{T})$ , *coefficient of a heat transfer*  $\mathbf{h}(\mathbf{T})$ ). In this case, heat conduction equation with non-linearly dependent thermo-physical parameters acquires the following form:

$$\rho \mathbf{c}(\mathbf{T}) \left( \frac{\partial \mathbf{T}}{\partial t} + u_x \frac{\partial \mathbf{T}}{\partial x} + u_y \frac{\partial \mathbf{T}}{\partial y} \right) = k_x(\mathbf{T}) \frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma \frac{\partial \mathbf{T}}{\partial x} \right) + k_y(\mathbf{T}) \frac{\partial^2 \mathbf{T}}{\partial y^2} + \mathbf{Q} \quad (8.43)$$

FEM is applied to the solution of the given equation; therefore the four-nodal isoparametric finite element is used (fig. 8.2). When fulfilling operations similar to those in section 8.3, we obtain the common system of nonlinear differential equations of the first order of the following form:

$$[\mathbf{M}(\mathbf{T})] \{\dot{\mathbf{T}}\} + [\mathbf{K}(\mathbf{T})] \{\mathbf{T}\} = \{\mathbf{P}(\mathbf{T}, t)\} \quad (8.44)$$

where matrices and vectors are defined similarly to section 8.3 but under condition that thermo-physical parameters  $\mathbf{c}$ ,  $\mathbf{k}_x$ ,  $\mathbf{k}_y$  and  $\mathbf{h}$  are functions of temperature.

*Specific heat and thermal conductivities* are described by polynomials of the second degree, namely:

$$\mathbf{c}(\mathbf{T}) = \mathbf{c}_0 + \mathbf{c}_1 \mathbf{T} + \mathbf{c}_2 \mathbf{T}^2 \quad (8.45)$$

$$\mathbf{k}(\mathbf{T}) = \mathbf{k}_0 + \mathbf{k}_1 \mathbf{T} + \mathbf{k}_2 \mathbf{T}^2, \quad (\mathbf{k} \in \mathbf{k}_x, \mathbf{k}_y) \quad (8.46)$$

The dependence of coefficient of heat transfer from temperature is described by a polynomial of the fourth degree of the following simple form:

$$\mathbf{h}(\mathbf{T}) = \mathbf{h}_0 + \mathbf{h}_1 \mathbf{T} + \mathbf{h}_2 \mathbf{T}^2 + \mathbf{h}_3 \mathbf{T}^3 + \mathbf{h}_4 \mathbf{T}^4 \quad (8.47)$$

Let us expand the vector's function:

$$\{\mathbf{R}(\mathbf{T})\} = [\mathbf{K}(\mathbf{T})] \{\mathbf{T}\} - \{\mathbf{P}(\mathbf{T}, t)\} \quad (8.48)$$

into the Taylor series on point  $\{\mathbf{T}\}_i$  and by keeping only the first two terms of the series, we obtain:

$$\{\mathbf{R}(\mathbf{T})\} \approx \{\mathbf{R}\}_i + [\mathbf{J}]_i \{\Delta \mathbf{T}\}_i = \{\mathbf{0}\} \quad (8.49)$$

$$[\mathbf{J}]_i = \frac{\partial \{\mathbf{R}\}_i}{\partial \{\mathbf{T}\}^T} - \text{Jacobi matrix and } \{\mathbf{R}\}_i = [\mathbf{K}(\mathbf{T})]_i \{\mathbf{T}\}_i - \{\mathbf{P}(\mathbf{T}, t)\} \quad (8.50)$$

Then equations system (8.44) acquire the following form:

$$[\mathbf{M}(\mathbf{T})] \{\dot{\mathbf{T}}\} + [\mathbf{J}]_i \{\Delta \mathbf{T}\}_i = -\{\mathbf{R}\}_i \quad (8.51)$$

By using the well known method of trapezoids, the vector  $\{\dot{\mathbf{T}}\}$  into an arbitrary moment of time  $\mathbf{t} + \boldsymbol{\tau}$  is represented as follows [53]:

$$\{\dot{T}_{t+\tau}\}_j = \frac{2}{\tau} (\{T_{t+\tau}\}_{j-1} + \{\Delta T\}_j - \{T_t\}) + \{\dot{T}_t\} \quad (8.52)$$

where  $\tau$  - integration step. In this case the equation (8.51) is changed by the finite-difference equation of the following form:

$$\left( \frac{2}{\tau} [M_{t+\tau}]_j + [J_{t+\tau}]_j \right) \{\Delta T\}_j = -\{R_{t+\tau}\}_j - [M_{t+\tau}]_j \left( \frac{2}{\tau} (\{T_{t+\tau}\}_j - \{T_t\}) + \{\dot{T}_t\} \right),$$

where  $\{\dot{T}_{t+\tau}\}_{j-1} = \frac{1}{\tau} (\{T_{t+\tau}\}_{j-1} - \{T_t\})$  (8.53)

Thus, the system of linear algebraic equations is solved on each step of  $i$ -iteration. The vector of temperatures into an arbitrary moment of time  $t+\tau$  is defined after each step of  $i$ -iteration by the following recursion relation:

$$\{T_{t+\tau}\}_j = \{T_{t+\tau}\}_{j-1} + \{\Delta T\}_j \quad (8.54)$$

The vector of velocity of change of temperature into an arbitrary moment of time  $t+\tau$  is defined by the following relation:

$$\{\dot{T}_{t+\tau}\}_j = \{\dot{T}_t\}_j + \frac{2}{\tau} (\{T_{t+\tau}\}_j - \{T_t\}) \quad (8.55)$$

The iterative process is being continued up to the realization of the following condition:

$$\max |\Delta T_j| \leq \varepsilon, \quad (j = 1, \dots, neq) \quad (8.56)$$

where  $\varepsilon$  - the given precision of solution and  $neq$  - total number of equations of the system.

#### 8.4.2. Input data for the solution of the problem

To solve the *nonlinear non-stationary problem of thermal conduction* in some plane, the program *Heat\_nonst\_nonlinear* is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order.

In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoin*), the number of groups of finite elements (*ngroup*) and the number of known temperatures (*nbond*) are coded. Those finite elements, which have identical *thermal conductivities*  $k_x$  and  $k_y$  are included into the corresponding group. In the *second* line, the number of the sides of finite elements, on which the coefficients of heat transfer (*ngradh*) are given, the heat flux (*ngradq*) and the number of finite elements, in which the interior sources of heat (*nq*) are known, are coded.

In the *third* line, the print code of intermediate results (*kprint*), the type of problem (*ngama*), the code of the system solution of algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of the system of equations system, precision of solution (*toler*) and the temperature of surrounding medium (*tapl*) are coded. If *kprint=0*, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* coordinates, *ngama=0*, otherwise - in *cylindrical* coordinates. The system of equations can be solved in two ways: if parameter *ksolve=0*, the *solve* function of *Maple-language* is used when solving the algebraic equations system; otherwise, the equations system is solved by the method of *conjugate gradients*.

In the *fourth* line, the number of integration steps (*ntime*), integration step (*dtime*) and velocities of body motion in the direction of axes  $X$  and  $Y$  (*velyx, veluy*) are coded. In the *fifth* line, the number of iterations (*niteration*) and the precision (*tol*) of solving nonlinear equations by Newton method are coded. In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of

this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – the number of nodes of the finite element, i.e. *nnode*=4} are coded, and also the number of the group, to which this finite element {array **Mgroup**(*nelem*)} belongs.

After arrays **Mtop** and **Mgroup** the elements of **Lbond**(*nbond*) array are coded line by line. The line number and the element of **Lbond** array is recorded into each line of the file. The numbers of nodes, in which the values of temperature are known, are recorded into each line of **Lbond** array. After of array **Lbond** the elements of **Lgradh**(*ngradh*, 3) array are coded line by line. The line number and elements of **Lgradh** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of the given element, on which the coefficients of heat transfer **h** are known, are coded into each line of **Lgradh** array.

After of array **Lgradh** the elements of **Lgradq**(*ngradq*, 3) array are coded line by line. The line number and elements of **Lgradq** array are recorded into each line of the file. The number of a finite element and two numbers of nodes of a side of this element, on which the heat flux **q** is known, are recorded into each line of **Lgradq** array. The elements of **Lq**(*nq*) array are coded line by line after of array **Lgradq**. The line number and the element of **Lq** array are recorded into each line of the file. The numbers of finite elements, in which the interior sources of heat **Q** are known, are recorded into each line of **Lq** array.

Behind array **Lq** file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line into the data. The **x**-coordinates and the **y**-coordinates of nodes of finite elements are recorded into the *first* and *second* columns of the **Coord** array. The number of a node as well as its (**x**,**y**)-coordinates are recorded into each line of the file. Behind **Coord** array the elements of **Param**(*ngroup*, 10) array are coded line by line. The line number of **Param** array, density of substance (**ρ**), the coefficients defining specific heat of substance (**c<sub>0</sub>**, **c<sub>1</sub>**, **c<sub>2</sub>**), the parameters defining thermal conductivities of substance in a direction of axes **X** and **Y** (**k<sub>x0</sub>**, **k<sub>x1</sub>**, **k<sub>x2</sub>**, **k<sub>y0</sub>**, **k<sub>y1</sub>**, **k<sub>y2</sub>**), are recorded into each line of the datafile. The line number of **Param** array should strictly correspond to the number of group of the used finite elements.

Behind array **Param**, the elements of **Storis**(*nelem*, 4) array are recorded line by line. The line number of **Storis** array and values of thickness of a body in each node of a finite element are recorded into each line of the datafile. If *axiall symmetric* problem of *thermal conduction* (*ngama*=1) is considered, the thickness of elements is not taken into account and the given array is not coded. Behind array **Storis**, the elements of **Bond**(*nbond*) array are recorded line by line. The line number of **Bond** array and the value of temperature in a node are recorded into each line of the datafile. The lines of **Bond** array should strictly correspond to the lines of **Lbond** array. After array **Bond** the elements of **Gradh**(*ngradh*, 5) arrays are recorded line by line. The line number of array **Gradh** and

values of **h<sub>i</sub>** (*i*=0 .. 4) coefficients defining the coefficient of heat transfer, i.e.: 
$$h(T) = \sum_{i=0}^4 h_i T^i$$
, are

recorded into each line. The lines of array **Gradh** should strictly correspond to the lines of array **Lgradh**.

After array **Gradh** the elements of **Gradq**(*ngradq*, 4) array are recorded line by line. The line number of **Gradq** array and values of **q<sub>i</sub>** (*i*=0..3) coefficients defining the heat flux, i.e. 
$$q(T) = [q_0 + q_1 \sin(q_2 t)] e^{q_3 t}$$
, are recorded into each line. The lines of **Gradq** array should correspond to the lines of **Lgradq** array. After array **Gradq** the elements of **Q**(*nq*) array are recorded line by line. The line number of **Q** array and the value of interior source of heat which is located in a finite element are recorded into each line. The lines of **Q** array should strictly correspond to the lines of **Lq** array.

If the boundary conditions are lacking, one element (*for example, 1*) is recorded only into array **Lbond**, while value 0 is ascribed to the element **Bond[1]** of the array. When there is lack of interior sources of heat into **Lq** array, only one element (*for example, 1*) is recorded; while 0-value is ascribed to element **Q[1]** of the array. In the datafile the lines of the comments with names of appropriate arrays are located between the recorded arrays. When reading the information from the datafile these text lines are skipped.

*Schematic structure of the datafile:*

Text line \*  
*nelem, npoin, ngroup, nbond, ngradh, ngradq, nq, kprint, ngama, ksolve, niter, toler, tapl, ntime, dtime, velux, veluy, niteration, tol*

Text line \*  
Arrays **Mtop(nelem, nnode), Mgroup(nelem)**

Text line \*  
Array **Lbond(nbond)**

Text line \*  
Array **Lgradh(ngradh, 3)**

Text line \*  
Array **Lgradq(ngradq, 3)**

Text line \*  
Array **Lq(nq)**

Text line \*  
Array **Coord(npoin, ndime)**

Text line \*  
Array **Param(ngroup, 10)**

Text line \*  
Array **Storis(nelem, 4)**

Text line \*  
Array **Bond(nbond)**

Text line \*  
Array **Gradh(ngradh, 5)**

Text line \*  
Array **Gradq(ngradq, 4)**

Text line \*  
Array **Q(nq)**

### 8.4.3. Brief description of Heat\_nonst\_nonlinear program solving the problem

The *Heat\_nonst\_nonlinear* program was programmed in Maple-language; it consists of the basic program and 39 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and the time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of temperature and velocity of its change in nodes of finite elements in the course of time. Calculation results are output on the monitor and are recorded into the file; hence a *qualifier* of a target file must be ascribed to variable *file\_rez1* of the program. The values of temperature and velocity of its change in the given nodes of finite elements are recorded into datafile *file\_rez1*.

### 8.4.4. An example of the use of Maple-program Heat\_nonst\_nonlinear

Allocation of temperature in a rectangular body consisting of four various materials is considered

as an important example: *copper (Cu)*, *aluminium (Al)*, insulating material – *asbestos-cement* and *steel* (fig. 8.9). On the right side, heat flux  $q$  is known, on the left side the coefficient of heat transfer  $h$  is given.

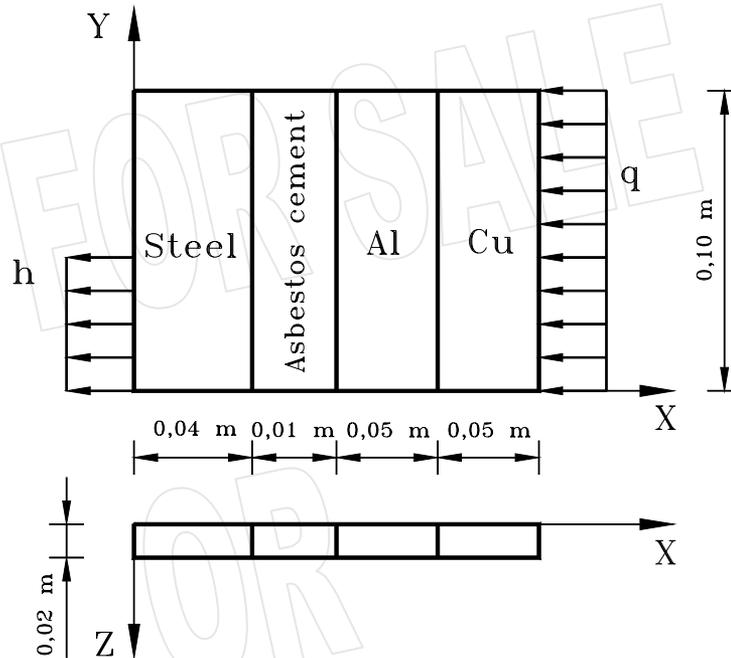


Fig. 8.9. The calculated scheme of the investigated body

It is supposed that the body moves in the direction of axis  $X$  with velocity 1 m/s. Hence, *density*, *specific heat* and *thermal conductivities* of materials are defined as follows:

$$\text{Cu: } \rho = 8933 \frac{\text{kg}}{\text{m}^3}; \quad c = 332,876 + 0,212054 \cdot T - 8,64772 \cdot 10^{-5} \cdot T^2 \frac{\text{J}}{\text{kg} \cdot \text{K}};$$

$$k_x = k_y = 428,774 - 0,0916488 \cdot T + 1,44768 \cdot 10^{-5} \cdot T^2 \frac{\text{J}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 1000 \text{ K};$$

$$\text{Al: } \rho = 2702 \frac{\text{kg}}{\text{m}^3}; \quad c = 661,53 + 0,813753 \cdot T - 2,71535 \cdot 10^{-4} \cdot T^2 \frac{\text{J}}{\text{kg} \cdot \text{K}}$$

$$k_x = k_y = 227,373 + 0,066194 \cdot T - 9,77612 \cdot 10^{-5} \cdot T^2 \frac{\text{W}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 800 \text{ K};$$

$$\text{Steel (Mn} \leq 1\%, 0.1\% \leq \text{Si} \leq 0.6\%): \rho = 7854 \frac{\text{kg}}{\text{m}^3};$$

$$c = 770,8551 - 1,59406 \cdot T + 1,96503 \cdot 10^{-3} \cdot T^2 \frac{\text{J}}{\text{kg} \cdot \text{K}};$$

$$k_x = k_y = 72,2536 - 0,0373551 \cdot T - 4,91872 \cdot 10^{-6} \cdot T^2 \frac{\text{W}}{\text{m} \cdot \text{K}};$$

$$200 \text{ K} \leq T \leq 1000 \text{ K};$$

$$\text{Asbestos-cement: } \rho = 1920 \frac{\text{kg}}{\text{m}^3}; \quad c = 1200 \frac{\text{J}}{\text{kg} \cdot \text{K}}; \quad k_x = k_y = 0,58 \frac{\text{W}}{\text{m} \cdot \text{K}}.$$

The coefficients for definition of heat transfer and heat flux are defined by the following relations accordingly:

$$h_0 = 40 \frac{W}{m^2 \cdot K} \quad h_1 = h_2 = h_3 = h_4 = 0$$

$$q_0 = -5 \cdot 10^6 \frac{W}{m^2} \quad q_1 = q_2 = q_3 = 0$$

Input data for the test example: Temperature of surrounding medium  $tapl=300$  K; thickness of body 0.002 m, the number of finite elements  $nelem=32$ , the number of nodes  $npoim=45$ , the number of groups of finite elements  $ngroup=4$ ,  $nbond=1$ ,  $ngradh=4$ ,  $kprint=0$ ,  $nq=1$ ,  $ngama=0$ ,  $niter=100$ ,  $ngradq=4$ ,  $ksolve=1$ ,  $niter=100$ ,  $toler=10^{-6}$ ,  $ntime=30$ ,  $dtime=10^{-3}$  s,  $velux=1$  m/s,  $veluy=0$ ,  $niteration=5$ ,  $tol=10^{-3}$ .

Results of calculation over the test example:

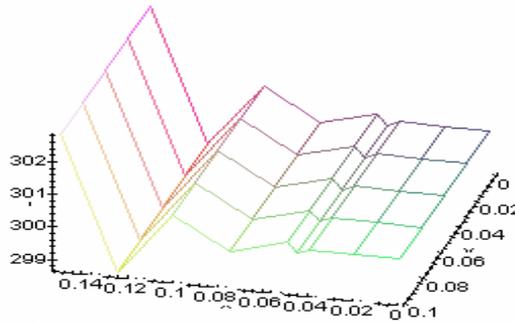
Values of temperature (*Temp*) and velocity of change of temperature (*Dtempdt*) in the nodes (*node*) of finite elements:

node=1	Temp=3.000036e+02	Dtempdt=3.750790e-01
node=2	Temp=2.999959e+02	Dtempdt=-3.732327e-01
node=3	Temp=3.000096e+02	Dtempdt=8.217162e-01
node=4	Temp=3.000036e+02	Dtempdt=3.750793e-01
node=5	Temp=2.999959e+02	Dtempdt=-3.732330e-01
node=6	Temp=3.000096e+02	Dtempdt=8.217166e-01
node=7	Temp=3.000036e+02	Dtempdt=3.750791e-01
node=8	Temp=2.999959e+02	Dtempdt=-3.732328e-01
node=9	Temp=3.000096e+02	Dtempdt=8.217163e-01
node=10	Temp=3.000036e+02	Dtempdt=3.750794e-01
node=11	Temp=2.999959e+02	Dtempdt=-3.732331e-01
node=12	Temp=3.000096e+02	Dtempdt=8.217167e-01
node=13	Temp=3.000036e+02	Dtempdt=3.750789e-01
node=14	Temp=2.999959e+02	Dtempdt=-3.732326e-01
node=15	Temp=3.000096e+02	Dtempdt=8.217160e-01
node=16	Temp=2.998618e+02	Dtempdt=-1.028464e+01
node=17	Temp=3.001654e+02	Dtempdt=9.510174e+00
node=18	Temp=2.998618e+02	Dtempdt=-1.028464e+01
node=19	Temp=3.001654e+02	Dtempdt=9.510174e+00
node=20	Temp=2.998618e+02	Dtempdt=-1.028464e+01
node=21	Temp=3.001654e+02	Dtempdt=9.510174e+00
node=22	Temp=2.998618e+02	Dtempdt=-1.028464e+01
node=23	Temp=3.001654e+02	Dtempdt=9.510174e+00
node=24	Temp=2.998618e+02	Dtempdt=-1.028464e+01
node=25	Temp=3.001654e+02	Dtempdt=9.510174e+00
node=26	Temp=2.997198e+02	Dtempdt=-1.484021e+01
node=27	Temp=3.006713e+02	Dtempdt=3.025587e+01
node=28	Temp=2.997198e+02	Dtempdt=-1.484021e+01
node=29	Temp=3.006713e+02	Dtempdt=3.025587e+01
node=30	Temp=2.997198e+02	Dtempdt=-1.484021e+01
node=31	Temp=3.006713e+02	Dtempdt=3.025587e+01
node=32	Temp=2.997198e+02	Dtempdt=-1.484021e+01
node=33	Temp=3.006713e+02	Dtempdt=3.025587e+01
node=34	Temp=2.997198e+02	Dtempdt=-1.484021e+01
node=35	Temp=3.006713e+02	Dtempdt=3.025587e+01
node=36	Temp=2.987303e+02	Dtempdt=-4.170074e+01
node=37	Temp=3.027805e+02	Dtempdt=5.480617e+01
node=38	Temp=2.987303e+02	Dtempdt=-4.170074e+01
node=39	Temp=3.027805e+02	Dtempdt=5.480617e+01

node=40	Temp=2.987303e+02	Dtempdt=-4.170074e+01
node=41	Temp=3.027805e+02	Dtempdt=5.480617e+01
node=42	Temp=2.987303e+02	Dtempdt=-4.170074e+01
node=43	Temp=3.027805e+02	Dtempdt=5.480617e+01
node=44	Temp=2.987303e+02	Dtempdt=-4.170074e+01
node=45	Temp=3.027805e+02	Dtempdt=5.480617e+01

Allocation of temperature in the considered body is represented on *fig. 8.10*.

**Temperature  $T(x,y)$  Distribution:**



**Fig. 8.10.** *The allocation of temperature in the considered body*

The *Heat\_nonst\_nonlinear* program is intended for the solution of *non-stationary nonlinear heat conduction equation* in an arbitrary flat body. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb5\_4\_new.mws*, *Mb5\_4.dat* and *Mb5\_4\_1.rez* accordingly.

# Chapter 9.

## Basic problems of the elasticity theory

In the present chapter in the brief form some problems of a linear mechanics of deformable bodies are considered. The basic relations concerning a description of a tensely-deformable state, and relations for elastic bodies in the elasticity theory are given. Each problem is solved by the FEM. The basic principles of the solution of the next practically important problems are represented:

- problem of definition of geometrical parameters of the cross-sections of bodies;
- calculation of a beam construction at static loadings;
- the plane problem of the elasticity theory;
- the contact problem of two elastic bodies.

The given problems represent the applied interest, therefore the *Maple* programs corresponding to them, are oriented onto a rather wide circle of the technical appendices.

This chapter presents research on the elasticity theory applying *finite elements*. The emphasis is on programming the *finite element method* in the *Maple* package to incorporate applications of the elasticity theory. The chapter contains research on the above problems on the elasticity theory. Versions of finite element programs in *Maple* is included in the attached archive. This material is directed to graduate students and researchers in civil, mechanical, aerospace engineering and materials science.

### 9.1. Definition of the geometrical characteristics of plane sections

The *resistance* of elements of constructions to various views of loadings depends not only on physical and mechanical properties of materials, but also from the geometrical form of their sections. For calculation of complicated constructions of mechanical engineering or separate details, the calculated scheme of which includes *beam elements*, it is necessary to know the geometrical characteristics of the cross-sections of beams (*coordinates of a gravity centre, area, axial moments of inertia etc.*). As a rule, at their definition of principal difficulties does not arise, but for sections of a composite configuration a volume of calculations and probability of arising of critical errors essentially increase. In this respect essential help to the constructor a computer-based analysis provides.

#### 9.1.1. Calculated expressions for definition of the geometrical characteristics of plane sections

The idea of definition of the geometrical characteristics of the plane sections consists in that what the cross-section is divided by some number of finite elements [74]. For increasing of precision of evaluation of the geometrical characteristics of a cross-section the two-dimensional *eight-nodal isoparametric finite element* is used (*fig. 9.1*). The given finite element precisely describes a contour of a cross-section of the second order.

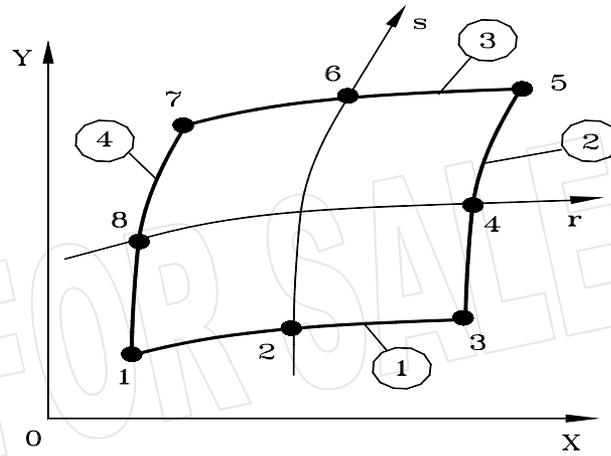


Fig. 9.1. Two-dimensional eight-nodal isoparametric finite element (the number in the circle indicates the number of the element side)

In each such finite element the local coordinate system  $\mathbf{r-s}$  is entered. The local coordinates of nodes of a finite element vary in the interval  $[-1, +1]$ . In some point of a plane section the global frame  $\mathbf{X-Y}$  is entered. Then the global coordinates of nodes can be approximated by the following relations:

$$\mathbf{x} = \sum_{i=1}^8 \mathbf{N}_i(\mathbf{r}, \mathbf{s}) \mathbf{X}_i = [\mathbf{N}(\mathbf{r}, \mathbf{s})] \{\mathbf{X}\} \quad (9.1)$$

$$\mathbf{y} = \sum_{i=1}^8 \mathbf{N}_i(\mathbf{r}, \mathbf{s}) \mathbf{Y}_i = [\mathbf{N}(\mathbf{r}, \mathbf{s})] \{\mathbf{Y}\} \quad (9.2)$$

where  $\mathbf{x}_i, \mathbf{y}_i$  - global coordinates of nodes and  $\mathbf{N}_i$  - shape functions, namely:

$$\begin{aligned} \mathbf{N}_1 &= \frac{1}{4}(1-r)(1-s) - \frac{1}{4}(1-s^2)(1-r) - \frac{1}{4}(1-r^2)(1-s); \\ \mathbf{N}_2 &= \frac{1}{2}(1-r^2)(1-s); \quad \mathbf{N}_4 = \frac{1}{2}(1+r)(1-s^2); \\ \mathbf{N}_3 &= \frac{1}{4}(1+r)(1-s) - \frac{1}{4}(1-r^2)(1-s) - \frac{1}{4}(1+r)(1-s^2); \\ \mathbf{N}_5 &= \frac{1}{4}(1+r)(1+s) - \frac{1}{4}(1-r^2)(1+s) - \frac{1}{4}(1+r)(1-s^2); \\ \mathbf{N}_6 &= \frac{1}{2}(1-r^2)(1+s); \quad \mathbf{N}_8 = \frac{1}{4}(1-r)(1-s^2); \\ \mathbf{N}_7 &= \frac{1}{4}(1-r)(1+s) - \frac{1}{4}(1-r^2)(1+s) - \frac{1}{4}(1-r)(1-s^2) \end{aligned} \quad (9.3)$$

The cross-sectional area is defined by the following relations:

$$\mathbf{A} = \sum_{i=1}^{\mathbf{NE}} \mathbf{A}^{(e)} = \sum_{i=1}^{\mathbf{NE}} \iint_{\mathbf{A}} dx \, dy = \sum_{i=1}^{\mathbf{NE}} \int_{-1}^{+1} \int_{-1}^{+1} \det[\mathbf{J}] \, dr \, ds, \quad (9.4)$$

where  $\mathbf{NE}$  - total number of the used finite elements and  $[\mathbf{J}]$  - functional Jacobi matrix of the following form:

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^8 \frac{\partial N_i}{\partial r} X_i & \sum_{i=1}^8 \frac{\partial N_i}{\partial r} Y_i \\ \sum_{i=1}^8 \frac{\partial N_i}{\partial s} X_i & \sum_{i=1}^8 \frac{\partial N_i}{\partial s} Y_i \end{bmatrix} \quad (9.5)$$

Static moments of a section concerning an axis are defined as follows:

$$S_x = \sum_{i=1}^{NE} \iint_A y \, dA = \sum_{i=1}^{NE} \iint_A y \, dx \, dy = \sum_{i=1}^{NE} \int_{-1}^{+1} \int_{-1}^{+1} y(r,s) \det[J] \, dr \, ds; \quad (9.6)$$

$$S_y = \sum_{i=1}^{NE} \iint_A x \, dA = \sum_{i=1}^{NE} \iint_A x \, dx \, dy = \sum_{i=1}^{NE} \int_{-1}^{+1} \int_{-1}^{+1} x(r,s) \det[J] \, dr \, ds \quad (9.7)$$

At the known static moments and the cross-sectional area the coordinates of a *gravity centre* are defined by the following obvious relations:

$$x_c = \frac{S_y}{A} \quad y_c = \frac{S_x}{A} \quad (9.8)$$

In this case the *axial moments* of inertia of a plane section are defined as follows:

$$I_x = \sum_{i=1}^{NE} I_{x,i} = \sum_{i=1}^{NE} \iint_{A_i} y^2 \, dA = \sum_{i=1}^{NE} \int_{-1}^{+1} \int_{-1}^{+1} y^2 \det[J] \, dr \, ds; \quad (9.9)$$

$$I_y = \sum_{i=1}^{NE} I_{y,i} = \sum_{i=1}^{NE} \iint_{A_i} x^2 \, dA = \sum_{i=1}^{NE} \int_{-1}^{+1} \int_{-1}^{+1} x^2 \det[J] \, dr \, ds \quad (9.10)$$

The *centrifugal moment of inertia* of a plane section concerning two perpendicular axes is defined by the following relation:

$$I_{xy} = \sum_{i=1}^{NE} I_{xy,i} = \sum_{i=1}^{NE} \iint_{A_i} xy \, dA = \sum_{i=1}^{NE} \int_{-1}^{+1} \int_{-1}^{+1} xy \det[J] \, dr \, ds \quad (9.11)$$

In these conditions the *axial central moments of inertia* are defined as follows:

$$I_{x_c} = I_x + y_c^2 A - 2y_c S_x; \quad (9.12)$$

$$I_{y_c} = I_y + x_c^2 A - 2x_c S_y \quad (9.13)$$

The *centrifugal central moment of inertia* is defined by the next relation:

$$I_{x_c y_c} = I_{xy} + x_c y_c A - x_c S_y - y_c S_x \quad (9.14)$$

While the *main central moments of inertia*  $I_{\max}$ ,  $I_{\min}$  receives the form:

$$I_{\max} = \frac{1}{2}(I_{x_c} + I_{y_c}) + \sqrt{\left(\frac{I_{x_c} - I_{y_c}}{2}\right)^2 + I_{x_c y_c}^2}; \quad (9.15)$$

$$I_{\min} = \frac{1}{2}(I_{x_c} + I_{y_c}) - \sqrt{\left(\frac{I_{x_c} - I_{y_c}}{2}\right)^2 + I_{x_c y_c}^2}. \quad (9.16)$$

We compute an *angle*, which defines standing of the main central axis  $\xi$ :

$$\alpha_{\xi} = \frac{1}{2} \operatorname{arctg}\left(\frac{I_{x_c y_c}}{I_{y_c} - I_{x_c}}\right) \quad (9.17)$$

Then the *minimal radius of inertia* is defined by the following relation:

$$R_{\min} = \sqrt{\frac{I_{\min}}{A}}. \quad (9.18)$$

In this case the approximate value of a *moment of inertia* at a *torsion* is equal:

$$I_p = \frac{4I_{x_c} I_{y_c}}{I_{x_c} + I_{y_c}}. \quad (9.19)$$

The moments of inertia relative to new axes  $X_1$ ,  $Y_1$ , passing via a beginning of the global frame (*point O*) and formative an angle  $\alpha$  between axis  $X$  and  $X_1$ , are defined by the following relations:

$$I_{x_1} = I_x \cos^2 \alpha + I_y \sin^2 \alpha - I_{xy} \sin(2\alpha); \quad (9.20)$$

$$I_{y_1} = I_x \sin^2 \alpha + I_y \cos^2 \alpha + I_{xy} \cos(2\alpha); \quad (9.21)$$

$$I_{x_1 y_1} = I_{xy} \cos \alpha + \frac{1}{2}(I_x - I_y) \sin(2\alpha) \quad (9.22)$$

### 9.1.2. Input data for the solution of the problem

For the solution of a problem of definition of the *geometrical characteristics of plane sections* the *Geometry* program is used. All data necessary for operation of this program, should be beforehand recorded into a file, for that to variable **F** the necessary qualifier of the file is ascribed. The data in the datafile are placed in the strict order, namely. In the *first* line the number of finite elements (*parameter nelem*) and number of nodes (*npoin*) are coded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop(nelem, nnode)**, where *nnode* - number of nodes of the finite element, i.e. *nnode*=8} are coded. After of array **Mtop** the elements of **Coord(npoin, ndime)** array, where *ndime* - dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, while into the *second* - the *y*-coordinates of nodes of finite elements are recorded. Into each line of the file the number of a node, and also its (*x,y*)-coordinates are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin*  
 Text line \*  
 Array *Mtop(nelem, mmode)*  
 Text line \*  
 Array *Coord(npoin, ndime)*

**9.1.3. Brief description of the Geometry program solving the problem**

The *Geometry* program has been programmed on the *Maple*-language; it consists of the basic program and 12 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates the *geometrical characteristics* of a *plane section* of the arbitrary form. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* the values of the geometrical characteristics of an explored plane section are recorded.

**9.1.4. An example of use of the Maple-program Geometry**

As an example of use of the program *Geometry* the geometrical characteristics of the plane cross-section represented on the *fig. 9.2*, are calculated.

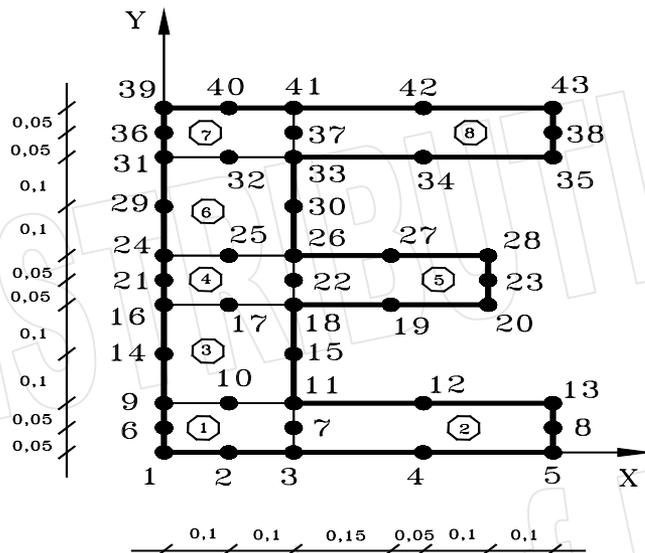


Fig. 9.2. Geometry of a plane cross-section

Input data for the test example: Number of finite elements *nelem*=8, number of nodes *npoin*=43.

Results of calculation over the test example:

Area=2.5e-01	
First Moment of area about X axes	Sx=8.75e-02
First Moment of area about Y axes	Sy=5.65e-02
Moment of inertia about X axes	Ix=4.363333e-02
Moment of inertia about Y axes	Iy=1.963333e-02
Polar moment of inertia	Iz=6.326666e-02
Area of product of inertia	Ixy=1.9775e-02
Centroid coordinate	Xc=2.26e-01

Centroid coordinate	Yc=3.5e-01
First Moment of area about centroid axes Xc	Sxc=0e-01
First Moment of area about centroid axes Yc	Syc=0e-01
Moment of inertia about centroid axes Xc	Ixc=1.300833e-02
Moment of inertia about centroid axes Yc	Iyc=6.864333e-03
Polar moment of inertia about centroid axes Zc	Izc=1.987266e-02
Area of product of inertia about centroid axes	Ixcyc=0e-01
Radius of gyration about centroid axes Xc	rxc=2.281081e-01
Radius of gyration about centroid axes Yc	ryc=1.657025e-01
Radius of gyration about centroid axes Zc	rzc=1.657025e-01

The *Geometry* program is intended for the calculation of the *geometrical characteristics* of a *plane section* of the arbitrary form. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb6\_1\_new.mws*, *Mb6\_1.dat* and *Mb6\_1\_1.rez* accordingly.

## 9.2. Computer-based calculation of the beam systems

The overwhelming majority of spatial engineering constructions contains the beam elements representing bodies, one dimensionality of which is essentially more than two others. The beams work onto *bending*, *tension* and *torsion*. For practical calculations of beams the following basic assumptions are quite valid, namely:

- the *hypothesis of plane sections*, according to which the cross-sections of a beam originally plane and normal ones to the beam axis, remain plane and normal ones to an elastic line of the beam and after its arcuation. Thus, the strain of an arcuation of a beam is considered irrespective of a shearing strain, which causes a standing of a plane of the cross-sections of the beam;
- it is supposed, that the longitudinal layers of a beam do not act one on another;
- the rather rigid beams are considered only, sags of which are small in comparison with height of a beam section, the angles of rotation of sections are small also.

The calculation of beam systems has major practical application in many practical problems, first of all, of a mechanical character. Computer-based facilities of calculation of similar systems in environment of the *Maple*-package here will be considered.

### 9.2.1. Calculated equations of the beam systems

As an example the beam element of length  $L$  with stiffness onto bending  $EI(x)$ , with stretching and squeezing  $EA(x)$  and coefficient of stiffness of the linear elastic basis  $k(x)$  is considered (fig. 9.3), where: accordingly  $E$ ,  $I(x)$ ,  $A(x)$  – *elastic module*, *axial moment of inertia* and *cross-sectional area* are as functions of  $x$ -coordinate [77]. For calculation of the *beam systems* the FEM is used. It is supposed, that the *cross-sectional area* of a finite element is approximated by a *quadratic polynominal* of the following simple form:

$$A(x) = A_0 + A_1x + A_2x^2, \quad (9.23)$$

where  $A_i$  – known coefficients ( $i=0 \dots 2$ ). The *axial moment of inertia* is approximated by a polynominal of the fourth degree of the following form:

$$I(x) = I_0 + I_1x + I_2x^2 + I_3x^3 + I_4x^4, \quad (9.24)$$

where  $I_i$  – known coefficients ( $i=0 \dots 4$ ).

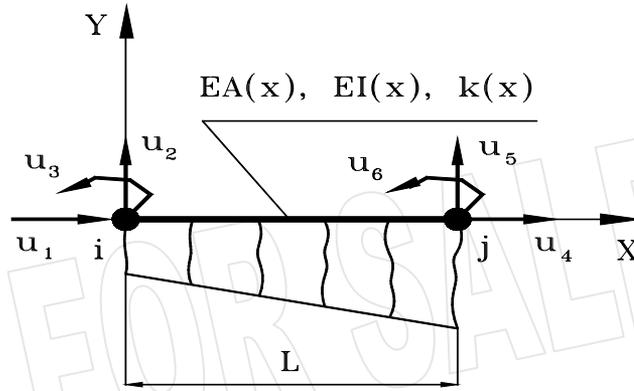


Fig. 9.3. A beam finite element and nodal displacements

A stretching and squeezing of an element is defined by displacements  $u_1$  and  $u_4$  along the  $X$ -axis, while the longitudinal displacements in a beam finite element are approximated by the following relation:

$$u(x) = N_1(x)u_1 + N_4(x)u_4, \quad (9.25)$$

where  $N_1(x) = 1 - \frac{x}{L}$ ;  $N_4(x) = \frac{x}{L}$  - shape functions of longitudinal displacements. Then the cross displacements in a beam finite element can be approximated as follows:

$$v(x) = N_2(x)u_2 + N_3(x)u_3 + N_5(x)u_5 + N_6(x)u_6, \quad (9.26)$$

where  $N_2(x) = 1 - 3\left(\frac{x}{L}\right)^2 + 2\left(\frac{x}{L}\right)^3$ ;  $N_3(x) = x\left[1 - \frac{2x}{L} + \left(\frac{x}{L}\right)^2\right]$ ;  $N_5(x) = 3\left(\frac{x}{L}\right)^2 - 2\left(\frac{x}{L}\right)^3$ ;

$$N_6(x) = x\left[-\frac{x}{L} + \left(\frac{x}{L}\right)^2\right]. \quad (9.27)$$

The coefficient of stiffness of the elastic basis is approximated by the following relation:

$$k(x) = N_1(x)k_i + N_4(x)k_j, \quad (9.28)$$

where  $k_i, k_j$  - coefficient of stiffness of the elastic basis in nodes  $i$  and  $j$  accordingly. In this case the potential energy of a beam element is defined as follows:

$$V = \frac{1}{2} \int_0^L EA(x) \left(\frac{\partial u}{\partial x}\right)^2 dx + \frac{1}{2} \int_0^L EI(x) \left(\frac{\partial^2 v}{\partial x^2}\right)^2 dx + \frac{1}{2} \int_0^L k(x) v(x)^2 dx. \quad (9.29)$$

The stiffness matrix of a beam element, which is obtained from expression of a potential energy, accepts the following form:

$$k_{ij} = \begin{cases} \int_0^L \frac{\partial^2 N_i}{\partial x^2} EI(x) \frac{\partial^2 N_j}{\partial x^2} dx + \int_0^L N_i k(x) N_j dx, & i, j = 2, 3, 5, 6 ; \\ \int_0^L \frac{dN_i}{dx} EA(x) \frac{dN_j}{dx} dx, & i, j = 1, 4 . \end{cases} \quad (9.30)$$

In our case the forces and distributed loading, which act onto the beam finite element, are represented on the fig. 9.4.

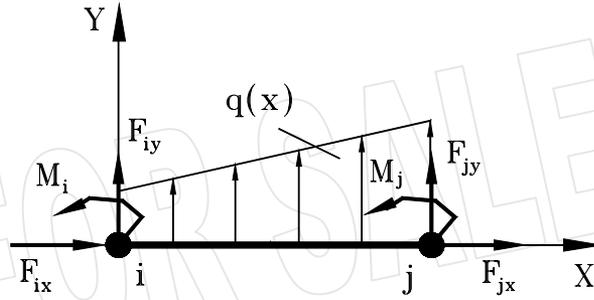


Fig. 9.4. The scheme of the loadings acting onto the beam finite element

The work of external forces acting onto the element, is defined by the following integral expression:

$$W = \int_0^L q(x) v(x) dx + \{\mathbf{u}\}^T \{\mathbf{F}\}, \quad (9.31)$$

where  $\{\mathbf{u}\}^T = \{u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6\}$  and  $\{\mathbf{F}\}^T = \{F_{ix} \ F_{iy} \ M_i \ F_{jx} \ F_{jy} \ M_j\}$  - a vector accordingly of nodal displacements and external forces acting onto the element. The vector of nodal loadings of a beam element being obtained from a work expression of external forces, is defined by the following system:

$$p_i = \begin{cases} \int_0^L q(x) N_i(x) dx + F_i, & i = 2, 3, 5, 6 ; \\ F_i, & i = 1, 4 . \end{cases} \quad (9.32)$$

Stiffness matrix and vector of nodal forces of all construction are formed of matrixes of stiffnesses and vectors of nodal forces of finite elements composed in a frame, which is common (global) for all construction. The transformation of a stiffness matrix and vector of nodal forces from a local frame into a common (global) frame by the following relations are provided:

$$[\mathbf{k}^{(e)}] = [\mathbf{T}]^T [\mathbf{k}^{(e)}] [\mathbf{T}]; \quad (9.33)$$

$$\{\mathbf{p}^{(e)}\} = [\mathbf{T}]^T \{\mathbf{p}^{(e)}\}, \quad (9.34)$$

where  $[\mathbf{k}^{(e)}]$ ,  $[\mathbf{k}^{(e)}]$  - global and local matrixes of stiffnesses of a finite e-element;  $\{\mathbf{p}^{(e)}\}$ ,  $\{\mathbf{p}^{(e)}\}$  - global and local vectors of nodal loadings of a finite e-element;  $[\mathbf{T}]$  - matrix of transformation of coordinates. In addition, the following relations are valid:

$$[\mathbf{T}] = \begin{bmatrix} [\lambda] & 0 \\ 0 & [\lambda] \end{bmatrix}; \quad [\lambda] = \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}; \quad c = \frac{X_j - X_i}{L}; \quad s = \frac{Y_j - Y_i}{L}, \quad (9.35)$$

where  $X_i, Y_i, X_j, Y_j$  - global coordinates of nodes i and j. For the solution of common equations system of the following form:

$$[\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{P}\} \quad (9.36)$$

it is necessary to define *boundary conditions* as follows:

$$\mathbf{u}_i = \mathbf{u}_{i_0} . \quad (9.37)$$

Then it is possible to solve the modified common equations system (9.36):

$$[\mathbf{K}_m]\{\mathbf{U}\} = \{\mathbf{P}_m\}, \quad (9.38)$$

where  $[\mathbf{K}_m]$ ,  $\{\mathbf{P}_m\}$  – modified stiffness matrix and vector of loadings. The reactions in bearings are defined by product of the unmodified stiffness matrix by a vector of nodal displacements:

$$\{\mathbf{R}\} = [\mathbf{K}]\{\mathbf{U}\}. \quad (9.39)$$

The *internal stresses* in a beam finite element in a local frame are defined by the following relation:

$$\{\mathbf{t}^{(e)}\} = [\mathbf{k}^{(e)}]\{\mathbf{u}^{(e)}\}. \quad (9.40)$$

### 9.2.2. Input data for the solution of the problem

For static calculation of the *plane beam systems* the *Strypas* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the datafile is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), number of groups of finite elements (*ngr*) and the number of known displacements (*nbond*) are coded. Into the corresponding group of the finite elements are included those elements, which have identical values of an *elastic modulus, area and moment of inertia* of a cross-section

In the *second* line, the number of the finite elements, which lie on the elastic basis (*nkd*), number of additional stiffnesses (*nst*), number of nodes, in which the external forces and moments (*nforc*) affect, and also number of finite elements onto which the distributed loadings affect (*nq*) are coded.

In the *third* line, a print code of intermediate results (*kprint*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*) are coded. If *kprint=0*, then the intermediate results are not printed, otherwise they are printed out. if parameter *ksolve=0*, then for the solution the *'solve'* function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem, nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode* = 2} are coded, and also the number of group, to which belongs this finite element {array **Mgr**(*nelem*)}. After of arrays **Mtop** and **Mgr** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and an element of **Lbond** array is recorded. Into the **Lbond** array the numbers of degrees of freedom with the given boundary conditions are recorded.

After of array **Lbond** the elements of **Lkd**(*nkd*) array are coded line by line. Into each line of the file the line number and elements of **Lkd** array are recorded. Into each line of **Lkd** array the numbers of finite elements which lie on the elastic basis, are coded. After of array **Lkd** the elements of **Lst**(*nst*) array are coded line by line. Into each line of the file the line number and elements of **Lst** array are recorded. Into each line of **Lst** array the numbers of finite elements, in which the additional stiffnesses are presetted, are recorded.

After of array **Lst** the elements of **Lforc**(*nforc*) array are coded line by line. Into each line of the file a line number and the element of **Lforc** array are recorded. Into each line of **Lforc** array the numbers of nodes, in which the external forces and moments operate, are recorded. After of array **Lforc** the

elements of **Lq(nq)** array are coded line by line. Into each line of the file a line number and the element of **Lq** array are recorded. Into each line of **Lq** array the number of a finite element, onto which the distributed loading affects, is recorded.

Behind of array **Lq** into the data file the elements of **Coord(npoin, ndime)** array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the **x**-coordinates, while into the *second* – the **y**-coordinates of nodes of finite elements are recorded. Into each line of the file the number of a node, and also its (**x,y**)-coordinates are recorded. Behind of array **Coord** the elements of **Bond(nbond)** array are recorded line by line. Into each line of the datafile a line number of **Bond** array and a value of boundary condition are recorded. The lines of **Bond** array should strictly correspond to the lines of **Lbond** array.

Behind of **Bond** array the elements of **Pgr(ngr, 9)** array are coded line by line. Into each line of the datafile a line number of **Pgr** array, value of an elastic modulus **E**, values of coefficients **A<sub>0</sub>**, **A<sub>2</sub>** and **I<sub>0</sub>**, **I<sub>1</sub>**, **I<sub>2</sub>**, **I<sub>3</sub>**, **I<sub>4</sub>**, by which are approximated a cross-section area and moment of section inertia are recorded. The line number of **Pgr** array should strictly correspond to the number of group of finite elements. After of array **Pgr** the elements of **Akd(nkd, 2)** array are recorded line by line. Into each line of the datafile, a line number of **Akd** array and values of coefficients of the elastic basis in each node of a finite element, are recorded. The lines of **Akd** array should correspond to the lines of **Lkd** array. After of array **Akd** the elements of **Ast(nst, 3)** array are recorded line by line. Into each line of the datafile, a line number of **Ast** array and the values of stiffness coefficients corresponding to each degree of freedom, are recorded. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After of array **Ast** the elements of **Forc(nforc, 3)** array are recorded line by line. Into each line, a line number of **Forc** array and the values of the external forces and moments, acting in the direction of each degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array. After of array **Forc** the elements of **Fq(nq, 2)** arrays are recorded line by line. Into each line, a line number of array **Fq** and the values of intensities of distributed loadings acting in each node of a finite element, are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nkd, nst, nforc, nq, kprint, ksolve, niter, toler*  
 Text line \*  
 Arrays **Mtop(nelem, nnode)**, **Mgr(nelem)**  
 Text line \*  
 Array **Lbond(nbond)**  
 Text line \*  
 Array **Lkd(nkd)**  
 Text line \*  
 Array **Lst(nst)**  
 Text line \*  
 Array **Lforc(nforc)**  
 Text line \*  
 Array **Lq(nq)**  
 Text line \*  
 Array **Coord(npoin, ndime)**

Text line \*  
 Array *Bond*(*nbond*)  
 Text line \*  
 Array *Pgr*(*ngr*, 9)  
 Text line \*  
 Array *Akd*(*nkd*, 2)  
 Text line \*  
 Array *Ast*(*nst*, 3)  
 Text line \*  
 Array *Forc*(*nforc*, 3)  
 Text line \*  
 Array *Fq*(*nq*, 2)

### 9.2.3. Brief description of the Strypas program solving the problem

The *Strypas* program was programmed on the *Maple*-language; it consists of the basic program and 17 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of *displacements of nodes*, the *internal stresses* in finite elements, and also *reactions in bearings*. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of *displacements of nodes*, the *internal stresses* in finite elements, and also *reactions in bearings* are recorded.

### 9.2.4. An example of use of the Maple-program Strypas

As an example of using of the program *Strypas* the beam construction, represented on the *fig. 9.5*, is considered. The given construction consists from three groups of finite elements: the first group - *horizontal beams*, the second group - *vertical beams*, and the third group - *inclined beams*.

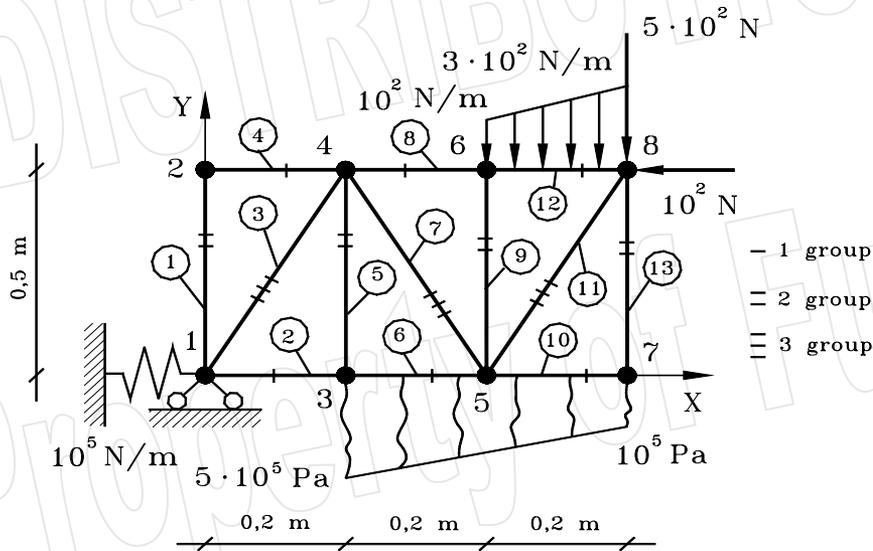


Fig. 9.5. Calculated scheme of the investigated beam construction

Input data for the test example: Number of the used finite elements *nelem*=13, number of nodes *npoin*=8, number of groups of finite elements *ngr*=3, number of boundary conditions *nbond*= 1,

$nkd=2, kprint=0, nq=1, nst=1, nforc=1, ksolve=0, niter=50, toler=10^{-6}$ .

Parameters of

the first group  $E = 2,10 \cdot 10^{11} \text{ Pa}; A_0 = 0,7853 \cdot 10^{-3} \text{ m}^2; A_1 = A_2 = 0;$   
 $I_0 = 0,4908 \cdot 10^{-5} \text{ m}^4; I_1 = I_2 = I_3 = I_4 = 0$

the second group  $E = 2,0 \cdot 10^{11} \text{ Pa}; A_0 = 0,7853 \cdot 10^{-3} \text{ m}^2; A_1 = -0,018849 \text{ m};$   
 $A_2 = 0,011309; I_0 = 0,49087 \cdot 10^{-5} \text{ m}^4; I_1 = -2,356 \cdot 10^{-5} \text{ m}^3;$   
 $I_2 = 4,241 \cdot 10^{-5} \text{ m}^2; I_3 = -3,3929 \cdot 10^{-5} \text{ m}; I_4 = 1,0178 \cdot 10^{-5}$

the third group  $E = 1,98 \cdot 10^{11} \text{ Pa}; A_0 = 3,1415 \cdot 10^{-4} \text{ m}^2; A_1 = A_2 = 0;$   
 $I_0 = 7,853 \cdot 10^{-9} \text{ m}^4; I_1 = I_2 = I_3 = I_4 = 0$

The remaining data necessary for calculation, are represented on the fig. 9.5.

Results of calculation over the test example:

The obtained displacements (**ux, uy, ua**) of nodes (**node**) of finite elements according to degrees of freedom:

node=1	ux=-1.000000e-03	uy=0e-01	ua=-1.522825e-02
node=2	ux=6.614606e-03	uy=1.441627e-07	ua=-1.522921e-02
node=3	ux=-1.000128e-03	uy=-3.045738e-03	ua=-1.522956e-02
node=4	ux=5.091662e-03	uy=-3.045719e-03	ua=-1.522960e-02
node=5	ux=-1.000253e-03	uy=-6.091938e-03	ua=-1.523190e-02
node=6	ux=6.614705e-03	uy=-6.091786e-03	ua=-1.523140e-02
node=7	ux=-1.000302e-03	uy=-9.138391e-03	ua=-1.523187e-02
node=8	ux=6.614649e-03	uy=-9.138239e-03	ua=-1.523232e-02

The calculated internal stresses in the finite elements has the following view:

Element=1		
Fx1=1.793433e+01	Fy1=-1.000000e+02	M1=0e-01
Fx2=-1.793433e+01	Fy2=0e-01	M2=0e-01
Element=2		
Fx1=6.260162e+00	Fy1=0e-01	M1=0e-01
Fx2=-1.062601e+02	Fy2=0e-01	M2=0e-01
Element=3		
Fx1=-3.638322e+01	Fy1=-1.005602e+02	M1=0e-01
Fx2=-8.338134e+00	Fy2=1.111751e+01	M2=0e-01
Element=4		
Fx1=4.793148e+00	Fy1=-6.390864e+00	M1=0e-01
Fx2=-4.793148e+00	Fy2=6.390864e+00	M2=0e-01
Element=5		
Fx1=7.382991e+00	Fy1=-2.948114e+04	M1=5.537046e+03
Fx2=-7.382991e+00	Fy2=2.948114e+04	M2=6.255409e+03
Element=6		
Fx1=1.029767e+02	Fy1=-1.715805e+02	M1=-5.888606e+00
Fx2=-1.029767e+02	Fy2=-1.837693e+02	M2=6.091756e+00
Element=7		
Fx1=1.942569e+01	Fy1=2.590093e+01	M1=0e-01
Fx2=-1.942569e+01	Fy2=-2.590093e+01	M2=0e-01
Element=8		
Fx1=-3.697037e+00	Fy1=-4.929382e+00	M1=0e-01

Fx2=3.697037e+00	Fy2=4.929382e+00	M2=0e-01
Element=9		
Fx1=4.773833e+01	Fy1=-3.145917e+05	M1=8.238519e+04
Fx2=-4.773833e+01	Fy2=3.145917e+05	M2=7.491067e+04
Element=10		
Fx1=4.004745e+01	Fy1=-3.665403e+01	M1=-1.441800e+00
Fx2=-4.004745e+01	Fy2=-6.589732e+01	M2=1.929188e+00
Element=11		
Fx1=-2.636547e+01	Fy1=2.510997e+01	M1=0e-01
Fx2=2.636547e+01	Fy2=-2.510997e+01	M2=0e-01
Element=12		
Fx1=4.630646e+01	Fy1=0e-01	M1=0e-01
Fx2=-4.630646e+01	Fy2=0e-01	M2=0e-01
Element=13		
Fx1=4.781330e+01	Fy1=-3.975656e+05	M1=9.599950e+04
Fx2=-4.781330e+01	Fy2=3.975656e+05	M2=1.027833e+05
Reaction in a bearing:	Number DOF=2	R=-9.790128e+01

The *Strypas* program is intended for static calculation of the plane beam systems of various destination. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in datafiles *Mb6\_2\_new.mws*, *Mb6\_2.dat* and *Mb6\_2\_1.rez* accordingly.

### 9.3. Plane problem of the elasticity theory

Some practically important *three-dimensional* problems of the elasticity theory can be reduced to *two-dimensional* plane problems, what essentially facilitates their solution. Depending on the geometrical sizes of bodies and a view of loadings, one may distinguish two types of plane problems of the elasticity theory - the *plane strain* and the *plane stress state*. The *plane strain* takes place in a medial part of rather long bodies in the direction of one of axes of coordinates weighted by stresses, perpendicular to this axis. Displacements along a long direction are neglectfully small, while others two in a plane, perpendicular to the axis of coordinates, directional along the long side, does not depend on coordinates in the direction of the given axis.

The *plane stress state* takes place in rather flat plates weighted by stresses, being in a plane of the plate. At the *plane tense state*, the stresses arising in the direction of an axis, perpendicular to a plane of a construction, are neglectfully small, while the remaining constituents of the stress does not depend on the given coordinate. In the present section the computer-based research techniques just of *plane problems* of the elasticity theory are considered.

#### 9.3.1. The calculated equations of a plane problem of the elasticity theory

The *total energy* of a *body deformation* with volume *V* is defined by the following integral:

$$\Pi = \frac{1}{2} \int_V (\{\boldsymbol{\varepsilon}\}^T - \{\boldsymbol{\varepsilon}_0\}) \{\boldsymbol{\sigma}\} dV \quad (9.41)$$

where  $\{\boldsymbol{\varepsilon}\}$ ,  $\{\boldsymbol{\varepsilon}_0\}$  - vectors of strain and initial strain accordingly;  $\{\boldsymbol{\sigma}\}$  - vector of stresses. The view of vector columns  $\{\boldsymbol{\varepsilon}\}$  and  $\{\boldsymbol{\sigma}\}$  depends on a type of the solved problem. In cartesian frame (*X*, *Y*) the given vector columns receives the next form:

$$\{\boldsymbol{\varepsilon}\}^T = [\boldsymbol{\varepsilon}_{xx} \quad \boldsymbol{\varepsilon}_{yy} \quad \boldsymbol{\varepsilon}_{xy}]; \quad (9.42)$$

$$\{\sigma\}^T = [\sigma_{xx} \quad \sigma_{yy} \quad \sigma_{xy}], \quad (9.43)$$

$$\text{where } \epsilon_{xx} = \frac{\partial u}{\partial x}; \quad \epsilon_{yy} = \frac{\partial v}{\partial y}; \quad \epsilon_{xy} = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} \quad (9.44)$$

In the *cylindrical frame*  $x=r, y=z$  (at the solution of a *axial-symmetric problem*) these vectors receive the following form:

$$\{\epsilon\}^T = [\epsilon_{xx} \quad \epsilon_{yy} \quad \epsilon_{xy} \quad \epsilon_{\theta\theta}]; \quad (9.45)$$

$$\{\sigma\}^T = [\sigma_{xx} \quad \sigma_{yy} \quad \sigma_{xy} \quad \sigma_{\theta\theta}], \quad (9.46)$$

$$\text{where } \epsilon_{xx} = \frac{\partial u}{\partial x}; \quad \epsilon_{yy} = \frac{\partial w}{\partial y}; \quad \epsilon_{xy} = \frac{\partial u}{\partial y} + \frac{\partial w}{\partial x}; \quad \epsilon_{\theta\theta} = \frac{u}{x} \quad (9.47)$$

For an elastic body takes place the Hooke law being expressed by the relation of the following form:

$$\{\sigma\} = [D](\{\epsilon\} - \{\epsilon_0\}), \quad (9.48)$$

where  $[D]$  – an *elasticity matrix* which has the following block structure:

$$[D] = \frac{E}{(1+\mu)(1-\mu_1)} \begin{bmatrix} 1 & \mu_1 & 0 \\ \mu_1 & 1 & 0 \\ 0 & 0 & \frac{1}{2}(1-\mu_1) \end{bmatrix}, \quad (9.49)$$

where:  $\mu_1 = \mu$  – for a *plane stress state*;  $\mu_1 = \frac{\mu}{1-\mu}$  – for a *plane strain*;  $\mu$  – the Poisson coefficient;  $E$  – the Young elastic modulus. For the *plane stress state* the stresses along the axis  $Z$  are equal:  $\sigma_{zz} = \sigma_{zx} = \sigma_{yz} = 0$ , while the *strain* along the axis  $Z$  is defined by the following relation:

$$\epsilon_{zz} = -\frac{\mu}{E}(\sigma_{xx} + \sigma_{yy}) = -\frac{\mu}{1-\mu}(\epsilon_{xx} + \epsilon_{yy}); \quad \epsilon_{yz} = \epsilon_{zx} = 0 \quad (9.50)$$

While for a *plane strain* the stresses along the axis  $Z$  are defined as follows:

$$\sigma_{zz} = \mu(\sigma_{xx} + \sigma_{yy}); \quad \sigma_{zx} = \sigma_{yz} = 0 \quad (9.51)$$

For the *axial-symmetric problem* the *elasticity matrix* accepts the next form:

$$[D] = \frac{E}{(1+\mu)(1-2\mu)} \begin{bmatrix} 1-\mu & \mu & 0 & \mu \\ \mu & 1-\mu & 0 & \mu \\ 0 & 0 & \frac{1-2\mu}{2} & 0 \\ \mu & \mu & 0 & 1-\mu \end{bmatrix} \quad (9.52)$$

The FEM is applied to the solution of a *plane problem* of the elasticity theory, for that the *quadrangular isoparametric finite element* is used, displacements in nodes of which the following simple *fig. 9.6* illustrates.

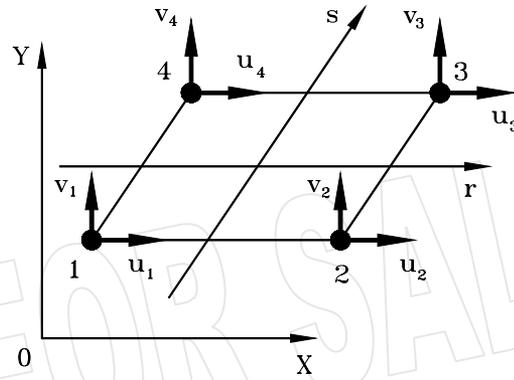


Fig. 9.6. Quadrangular isoparametric finite element

The displacement vector  $\{u\}$  in such finite element is approximated as follows:

$$\{u(r, s)\} = \sum_{i=1}^4 N_i(r, s) u_i = [N(r, s)] \{u^{(e)}\}, \quad (9.53)$$

where  $[N(r, s)]$  - matrix of shape functions (see section 9.2) and  $\{u^{(e)}\}$  - vector of nodal displacements. By using expressions (9.44) and (9.47), the vector of strains  $\{\epsilon\}$  can be expressed via nodal displacements  $\{u^{(e)}\}$ , namely:

$$\{\epsilon\} = [B] \{u^{(e)}\}. \quad (9.54)$$

The matrix  $[B]$  is obtained at differentiation of expression (9.53) and consists from blocks, amount of which is equal to number of nodes of an element, i.e.:  $[B] = [B_1 \ B_2 \ B_3 \ B_4]$ . For various coordinate systems the given blocks have the following form accordingly:

for the cartesian frame:  $[B_i] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \end{bmatrix}, \quad (9.55)$

for the cylindrical frame ( $x = r, y = z$ ):  $[B_i] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} \\ \frac{N_i}{x} & 0 \end{bmatrix}. \quad (9.56)$

The energy of strain of an finite e-element is defined from the next relation:

$$\Pi^{(e)} = \frac{1}{2} \{u^{(e)}\}^T [k^{(e)}] \{u^{(e)}\} - \{u^{(e)}\}^T \{p_\epsilon^{(e)}\}, \quad (9.57)$$

where  $[k^{(e)}]$  - stiffness matrix of a finite element, defined by the next relations:

$$[\mathbf{k}^{(e)}] = \int_{V^{(e)}} [\mathbf{B}]^T [\mathbf{D}] [\mathbf{B}] dV; \quad (9.58)$$

$$\{\mathbf{p}_\varepsilon^{(e)}\} = \int_{V^{(e)}} [\mathbf{B}]^T [\mathbf{D}] \{\boldsymbol{\varepsilon}_0\} dV. \quad (9.59)$$

The work of external forces acting onto a finite  $e$ -element, is expressed as:

$$\mathbf{W}^{(e)} = \int_{V^{(e)}} \{\mathbf{u}^{(e)}\}^T \{\mathbf{f}_V^{(e)}\} dV + \int_{S^{(e)}} \{\mathbf{u}^{(e)}\}^T \{\mathbf{f}_S^{(e)}\} dS + \{\mathbf{u}^{(e)}\}^T \{\mathbf{f}_k^{(e)}\} \quad (9.60)$$

where  $\{\mathbf{f}_V^{(e)}\}$ ,  $\{\mathbf{f}_S^{(e)}\}$ ,  $\{\mathbf{f}_k^{(e)}\}$  - vectors of volumetric, of a surface and concentrated forces accordingly. Then, by minimizing a complete potential energy of a finite element, we obtain the following equations system:

$$[\mathbf{k}^{(e)}] \{\mathbf{u}^{(e)}\} = \{\mathbf{p}^{(e)}\}, \quad (9.61)$$

$$\text{where } \{\mathbf{p}^{(e)}\} = \{\mathbf{p}_V^{(e)}\} + \{\mathbf{p}_S^{(e)}\} + \{\mathbf{p}_\varepsilon^{(e)}\} + \{\mathbf{f}_k^{(e)}\}; \quad (9.62)$$

$$\{\mathbf{p}_V^{(e)}\} = \int_{V^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_V^{(e)}\} dV; \quad (9.63)$$

$$\{\mathbf{p}_S^{(e)}\} = \int_{S^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_S^{(e)}\} dS. \quad (9.64)$$

In addition, the elementary volume  $dV$  is defined by the following relation:

$$dV = \Delta(\mathbf{r}, \mathbf{s})^{1-\gamma} \mathbf{x}^\gamma \det[\mathbf{J}] d\mathbf{r} d\mathbf{s}, \quad (9.65)$$

where  $\Delta(\mathbf{r}, \mathbf{s})$  - thickness of a finite element,  $\Delta(\mathbf{r}, \mathbf{s}) = [\mathbf{N}(\mathbf{r}, \mathbf{s})] \{\Delta\}$ ,  $\{\Delta\}^T = [\Delta_1 \ \Delta_2 \ \Delta_3 \ \Delta_4]$ ;  $\gamma = 0$  at use of the cartesian frame and  $\gamma = 1$  at use of the cylindrical frame for axial-symmetric problem ( $\mathbf{x} = r$ ,  $\mathbf{y} = z$ );  $[\mathbf{J}]$  - Jacobi matrix of the following form:

$$[\mathbf{J}] = \begin{bmatrix} \sum_{i=1}^4 \frac{\partial N_i}{\partial r} X_i & \sum_{i=1}^4 \frac{\partial N_i}{\partial r} Y_i \\ \sum_{i=1}^4 \frac{\partial N_i}{\partial s} X_i & \sum_{i=1}^4 \frac{\partial N_i}{\partial s} Y_i \end{bmatrix}, \quad (9.66)$$

where:  $X_i$ ,  $Y_i$  - global coordinates of nodes of an explored finite element. Then the elementary surface area  $dA$  is defined by relation of the following form:

$$dA = \Delta(\mathbf{r}, \mathbf{s})^{1-\gamma} \mathbf{x}^\gamma d\Gamma \quad (9.67)$$

where  $d\Gamma = \sqrt{\left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2} d\mathbf{r}$  for the sides 1 and 3, and  $d\Gamma = \sqrt{\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2} d\mathbf{s}$  for the sides 2

and **4** of the explored finite element. By uniting the matrixes of stiffnesses and the vectors of loadings of all finite elements, we obtain a global equations system of the following form:

$$[\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{P}\} \quad (9.68)$$

For the solution of the given equations system it is necessary to define boundary conditions:  $\mathbf{u}_i = \mathbf{u}_0$ . In addition, the stresses in an finite **e**-element are defined as follows:

$$\{\boldsymbol{\sigma}^{(e)}\} = [\mathbf{D}][\mathbf{B}]\{\mathbf{u}^{(e)}\} = [\mathbf{S}]\{\mathbf{u}^{(e)}\} \quad (9.69)$$

The *tense state* is characterized by equivalent stresses, which are defined by the following relation:

$$\sigma_{ekv} = \sqrt{\frac{1}{2}(\mathbf{S}_x^2 + \mathbf{S}_y^2 + \mathbf{S}_z^2) + \sigma_{xy}^2} \quad (9.70)$$

where  $\mathbf{S}_{ij}$  – *deviator stresses* being defined by the following relations:

$$\mathbf{S}_x = \sigma_{xx} - \sigma_m; \mathbf{S}_y = \sigma_{yy} - \sigma_m; \mathbf{S}_z = \sigma_{zz} - \sigma_m; \sigma_m = \frac{1}{3}(\sigma_{xx} + \sigma_{yy} + \sigma_{zz}) \quad (9.71)$$

The obtained relations underlie of evaluation of values of displacements, strains and stresses in nodes of finite elements, and also reactions in bearings.

### 9.3.2. Input data for the solution of the problem

For the solution of *plane problem* of the *elasticity theory* the *Plane* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), number of groups of finite elements (*ngr*) and the number of known displacements (*nbond*) are coded. Into the corresponding group of the finite elements are included those elements, which have identical values of an *elastic modulus* and the *Poisson coefficient*.

In the *second* line, the number of nodes, in which the additional stiffnesses (*nst*) are known, number of nodes, to which the external forces (*nforc*) are applied, number of the sides of finite elements, onto which the distributed loadings (*nq*) are acted, and a code of the plane tense state (*kstr*), are recorded; *kstr=0* for the *plane stress*, and *kstr=1* for the *plane strain*.

In the *third* line, a print code of intermediate results (*kprint*), a problem type (*ngama*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, a precision of the solution (*toler*) and density of a material (*tankis*) are coded. If *kprint=0*, then the intermediate results are not printed; otherwise, they are printed out. If the problem is solved in the *cartesian* frame, then *ngama=0*, if *ngama=1* – that in the *cylindrical* frame. if parameter *ksolve=0*, then for the solution the *solve* function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode=4*} are coded, and also the number of group, to which belongs this finite element {array **Mgr**(*nelem*)}. After of arrays **Mtop** and **Mgr** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and an element of **Lbond** array is recorded. Into the **Lbond** array the numbers of degrees of freedom with the given boundary conditions are recorded.

After of array **Lbond** the elements of **Lst**(*nst*) array are coded line by line. Into each line of the file the line number and elements of **Lst** array are recorded. Into each line of **Lst** array the numbers of nodes, in which the additional stiffnesses are presettet, are recorded. After of array **Lst** the

elements of **Lforc**(*nforc*) array are coded line by line. Into each line of the file a line number and the element of **Lforc** array are recorded. Into each line of **Lforc** array the numbers of nodes, in which the external forces operate, are recorded.

After of array **Lforc** the elements of **Lq**(*nq*) array are coded line by line. Into each line of the file a line number and the element of **Lq** array are recorded. Into each line of **Lq** array the number of a finite element and the numbers of nodes located on a side of a finite element, onto which the distributed loading affects, is recorded.

Behind of array **Lq** into the data file the elements of **Coord**(*npoint*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the first column of the **Coord** array the *x*-coordinates, while into the second – the *y*-coordinates of nodes of finite elements are recorded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded. Behind of array **Coord** the elements of **Bond**(*nbond*) array are recorded line by line. Into each line of the datafile a line number of **Bond** array and a value of displacement are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

Behind of **Bond** array the elements of **Pgr**(*ngr*, 2) array are coded line by line. Into each line of the datafile a line number of **Pgr** array, value of an elastic modulus and the Poisson coefficient of the corresponding group are recorded. The line number of **Pgr** array should strictly correspond to the number of group of finite elements.

After of array **Pgr** the elements of **Storis**(*nelem*, *nnode*) array are recorded line by line. Into each line of the datafile, a line number of **Storis** array and values of thickness of an element in each node, are recorded. In case of the *axial-symmetric* problem (*ngama*=1) thickness of elements is not considered and the given array is not formed. After of array **Storis** the elements of **Ast**(*nst*, 2) array are recorded line by line. Into each line of the datafile, a line number of **Ast** array and the values of the stiffnesses corresponding to each degree of freedom in the given node, are coded. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After of array **Ast** the elements of **Forc**(*nforc*, 2) array are recorded line by line. Into each line, a line number of **Forc** array and the values of external forces acting in a direction of each degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array. After of array **Forc** the elements of **Fq**(*nq*, 2) arrays are recorded line by line. Into each line, a line number of array **Fq** and the values of intensities of distributed loadings acting onto a side of a finite element in its two nodes, are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nst, nforc, nq, kstr, kprint, ngama, ksolve, niter, toler, tankis*  
Text line \*  
Arrays **Mtop**(*nelem*, *nnode*), **Mgr**(*nelem*)  
Text line \*  
Array **Lbond**(*nbond*)  
Text line \*  
Array **Lst**(*nst*)  
Text line \*  
Array **Lforc**(*nforc*)  
Text line \*  
Array **Lq**(*nq*, 3)

Text line \*  
 Array *Coord*(*npoin*, *ndime*)  
 Text line \*  
 Array *Bond*(*nbond*)  
 Text line \*  
 Array *Pgr*(*ngr*, 2)  
 Text line \*  
 Array *Storis*(*nelem*, *nnode*)  
 Text line \*  
 Array *Ast*(*nst*, 2)  
 Text line \*  
 Array *Forc*(*nforc*, 2)  
 Text line \*  
 Array *Fq*(*nq*, 2)

### 9.3.3. Brief description of the Plane program solving the problem

The *Plane* program was programmed on the *Maple*-language; it consists of the basic program and 42 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of *displacements*, the *strains* and *stresses* in nodes of finite elements, and also *reactions in bearings*. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of *displacements*, the *strains* and *stresses* in nodes of finite elements, and also *reactions in bearings* are recorded.

### 9.3.4. An example of use of the Maple-program Plane

As an application example for the *Plane* program, the ring consisting of two stratum of materials with the square cross-section is considered, namely: *carbon steel* and *aluminium* (fig. 9.7). In addition, from above onto the ring a distributed loading affectes, and in the inferior right corner the stiffnesses of bearings are known.

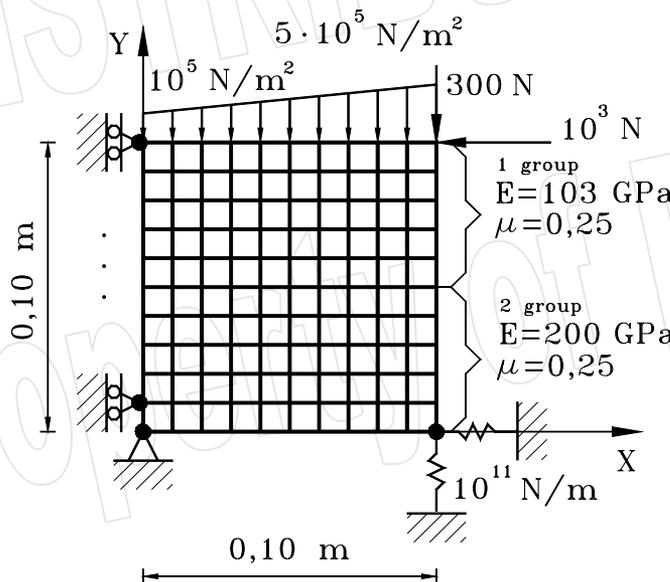


Fig. 9.7. The calculated scheme of the investigated problem

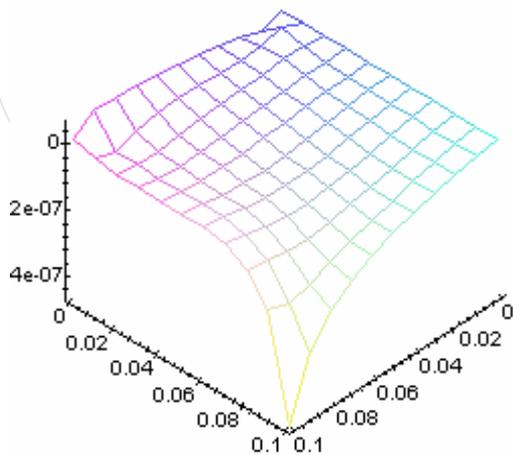
*Input data for the test example:* number of the finite elements  $n_{elem}=100$ , number of nodes  $n_{poin}=121$ , number of groups of finite elements  $n_{gr}=2$ , number of boundary conditions  $n_{bond}=12$ ,  $k_{str}=0$ ,  $k_{print}=0$ ,  $n_q=10$ ,  $n_{st}=1$ ,  $n_{forc}=1$ ,  $k_{solve}=1$ ,  $n_{iter}=500$ ,  $t_{oler}=10^{-6}$ ,  $n_{gama}=1$ ,  $t_{ankis}=7850 \text{ kg/m}^3$ .

**Results of calculation over the test example:**

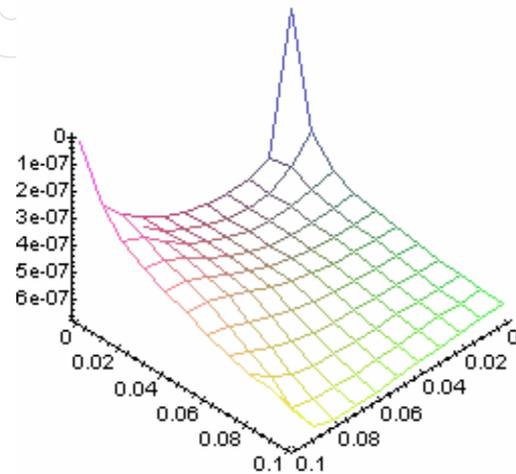
The displacements of nodes of finite elements of the cross-section of the ring in the direction of axes X (a) and Y (b) are represented on the fig. 9.8.

**Displacement  $U_x(x,y)$  Distribution:**

**Displacement  $U_y(x,y)$  Distribution:**



(a)

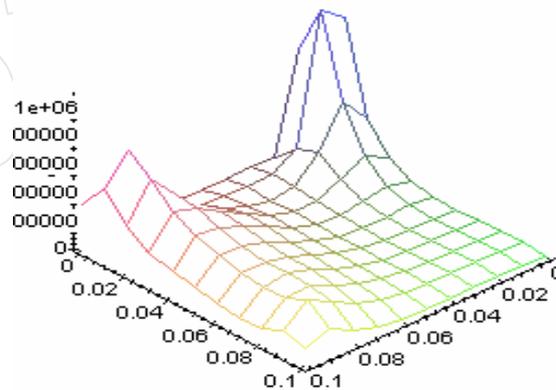


(b)

**Fig. 9.8.** Displacements of nodes of finite elements of cross-section of the ring

While the allocation of the equivalent stresses in the cross-section of the ring is represented on fig. 9.9.

**Ekv. Stresses  $S(x,y)$  Distribution:**



**Fig. 9.9.** Allocation of the equivalent stresses in cross-section of the ring

In addition, the reactions (R) in bearings (DOF) are defined by the following values:

Number DOF=1	R=6.863854e+00
Number DOF=2	R=1.431630e+02
Number DOF=23	R=-5.260766e+01
Number DOF=45	R=-6.834721e+00
Number DOF=67	R=-9.209422e-01
Number DOF=89	R=-1.638256e+00

Number DOF=111	R=-1.850991e+00
Number DOF=133	R=-1.774817e+00
Number DOF=155	R=-2.162124e+00
Number DOF=177	R=-2.412828e+00
Number DOF=199	R=-2.439529e+00
Number DOF=221	R=-1.214807e+00

The *Plane* program is intended for the solution of a *plane problem* of the elasticity theory. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles *Mb6\_3\_new.mws*, *Mb6\_3.dat* and *Mb6\_3\_1.rez* accordingly.

## 9.4. Problem of contact of the elastic bodies

The *reliability* and *longevity* of machine-building constructions is substantially defined by efficiency of *contacting bodies* composing them. The contact stiffness exert major influence onto their common stiffness. In turn, the *contact stiffness* depends on geometry of contact, and also of physical and mechanical properties of contacting bodies. The field of contact of deformable bodies represents a surface, on which the contact efforts operate. The geometry of a contact surface generally represents a rather composite configuration. Depending on geometry of a contact surface, of the acting loadings onto contacting bodies, and also of physical and mechanical properties of these bodies in the field of contact can arise phenomena such as: *tripping*, *slide* and *tearing off* of the contacting surfaces. Moreover, in some field of contact can arise and *plastic strains*. In the present section a *contact problem* of *elastic bodies without phenomenon of a slippage* is considered. The solution of the problem is founded on the principle of minimum of potential energy and is reduced to a problem of minimization of a quadratic functional with linear limitations obtained from geometrical conditions of the contact. For the numerical solution of the problem the FEM allowing to take into account the features of *geometry* of contacting bodies and also an *tense state* at an arbitrary loading is used.

### 9.4.1. The calculated equations of the problem of contact of elastic bodies

The problem of contact of elastic bodies is considered in some plane at the supposition that in *unstrained state* the bodies adjoin in points, whose *form* and *sizes* are defined from geometrical reasons. For the solution of the given problem the quadrangular isoparametric finite element represented on the *fig. 9.6*, is used. In this case a surface of contacting bodies consists from the next three parts, namely:  $\mathbf{S} = \mathbf{S}_u + \mathbf{S}_\sigma + \mathbf{S}_c$ , where:  $\mathbf{S}_u$  – part of a surface, on which the displacements  $u$  are presetted;  $\mathbf{S}_\sigma$  – part of a surface, on which the surface strains are presetted;  $\mathbf{S}_c$  – zone of contact of bodies  $\Omega_1$  and  $\Omega_2$ . The boundary conditions on the surface  $\mathbf{S}_c$  represent conditions of *unpenetration*. While the condition of contact of two finite elements of the bodies  $\Omega_1$  and  $\Omega_2$  on boundary of  $\mathbf{S}_c$  has the following form (*fig. 9.10*), namely.

The boundary conditions on the surface represent conditions of *unpenetration*. While the condition of *contact* of two finite elements of the bodies and on the *boundary* has the following form (*fig. 9.10*):

$$\{\mathbf{a}_k\} = \{\mathbf{u}_{k,2}\} - \{\mathbf{u}_{k,1}\} + \{\Delta_k\} = \{\mathbf{0}\} \quad (9.72)$$

where  $\{\mathbf{u}_{k,j}\}$  ( $j = 1, 2$ ) – vectors of displacements of the contacting nodes of two finite elements, namely:

$$\{\mathbf{u}_{k,j}\}^T = [u_{k,j,n,i} \quad u_{k,j,\tau,i} \quad u_{k,j,n,i+1} \quad u_{k,j,\tau,i+1}] \quad (9.73)$$



$$\begin{aligned} \cos \alpha_i &= \frac{(x_{i+1} - x_i)}{L_{i-1,i+1}}, \quad \sin \alpha_i = \frac{(y_{i+1} - y_i)}{L_{i-1,i+1}}, \quad i = 2, \dots, n_k - 1; \quad \cos \alpha_i = \frac{(x_{i+1} - x_i)}{L_{i+1,i}}, \\ \sin \alpha_i &= \frac{(y_{i+1} - y_i)}{L_{i+1,i}}, \quad i = 1; \quad \cos \alpha_i = \frac{(x_{i+1} - x_i)}{L_{i-1,i+1}}, \quad \sin \alpha_i = \frac{(y_{i+1} - y_i)}{L_{i-1,i+1}}, \quad i = n_k, \\ L_{i,j} &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}; \quad n_k - \text{total number of nodes in the contact} \end{aligned}$$

By substituting the expression (9.76) into (9.75), we obtain the following relation:

$$\Pi_k^{(e)} = \frac{1}{2} \{ \mathbf{u}_{k,1}^{(e)} \}^T [ \mathbf{k}_{k,1}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} + \frac{1}{2} \{ \mathbf{u}_{k,2}^{(e)} \}^T [ \mathbf{k}_{k,2}^{(e)} ] \{ \mathbf{u}_{k,2}^{(e)} \} - \{ \mathbf{u}_{k,1}^{(e)} \}^T [ \mathbf{p}_{k,1}^{(e)} ] - \{ \mathbf{u}_{k,2}^{(e)} \}^T [ \mathbf{p}_{k,2}^{(e)} ] - \int_{S_c^{(e)}} \{ \mathbf{q}_k \}^T \{ \mathbf{a}_k \} dS, \quad (9.77)$$

$$\text{where } [ \mathbf{k}_{k,j}^{(e)} ] = [ \mathbf{T} ]^T [ \mathbf{k}_j^{(e)} ] [ \mathbf{T} ]; \quad [ \mathbf{p}_{k,j}^{(e)} ] = [ \mathbf{T} ]^T [ \mathbf{p}_j^{(e)} ]; \quad j = 1, 2. \quad (9.78)$$

The vector of an initial gap  $\{ \Delta_k \}$ , vector of contact efforts  $\{ \mathbf{q}_k \}$  and vectors of contact displacements  $\{ \mathbf{u}_{k,j} \}$  can be expressed via a vector of an initial gap, vector of contact efforts and vector of contact displacements of a finite element as follows:

$$\{ \Delta_k \} = [ \mathbf{A} ] \{ \Delta^{(e)} \}; \quad \{ \mathbf{q}_k \} = [ \mathbf{B} ] \{ \mathbf{q}_k^{(e)} \}; \quad \{ \mathbf{u}_{k,j} \} = [ \mathbf{C} ] \{ \mathbf{u}_k^{(e)} \} \quad (9.79)$$

where  $[ \mathbf{A} ]$ ,  $[ \mathbf{B} ]$ ,  $[ \mathbf{C} ]$  – matrixes of the state from approximating functions  $\mathbf{N}_i(\mathbf{r}, \mathbf{s})$ .

Then, the integral in the expression (9.77) accepts the following form:

$$\int_{S_c^{(e)}} \{ \mathbf{q}_k \}^T \{ \mathbf{a}_k \} dS = \{ \mathbf{q}_k^{(e)} \}^T \left( [ \mathbf{A}_{k,2}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} - [ \mathbf{A}_{k,1}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} - \{ \mathbf{f}^{(e)} \} \right), \quad (9.80)$$

$$\text{where } [ \mathbf{A}_{k,j}^{(e)} ] = \int_{S_c^{(e)}} [ \mathbf{B} ]^T [ \mathbf{C}_2 ] dS, \quad j = 1, 2 \quad \text{and} \quad \{ \mathbf{f}^{(e)} \} = \int_{S_c^{(e)}} [ \mathbf{B} ]^T [ \mathbf{A} ] \{ \Delta^{(e)} \} dS$$

Then the potential energy of two contacting finite elements is equal:

$$\Pi_k^{(e)} = \frac{1}{2} \sum_{j=1}^2 \left( \{ \mathbf{u}_{k,j}^{(e)} \}^T [ \mathbf{k}_{k,j}^{(e)} ] \{ \mathbf{u}_{k,j}^{(e)} \} - \{ \mathbf{u}_{k,j}^{(e)} \}^T [ \mathbf{p}_{k,j}^{(e)} ] \right) + \{ \mathbf{q}_k^{(e)} \}^T \left( [ \mathbf{A}_{k,2}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} - [ \mathbf{A}_{k,1}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} - \{ \mathbf{f}^{(e)} \} \right) \quad (9.81)$$

By minimizing the potential energy of two contacting finite elements, we obtain an equations system of a standing balance of finite elements and also condition of compatibility of displacements on a surface of the contact, namely:

$$[ \mathbf{k}_{k,1}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} = [ \mathbf{p}_{k,1}^{(e)} ] + [ \mathbf{A}_{k,1}^{(e)} ]^T \{ \mathbf{q}_k^{(e)} \}; \quad (9.82)$$

$$[ \mathbf{k}_{k,2}^{(e)} ] \{ \mathbf{u}_{k,2}^{(e)} \} = [ \mathbf{p}_{k,2}^{(e)} ] - [ \mathbf{A}_{k,2}^{(e)} ]^T \{ \mathbf{q}_k^{(e)} \}; \quad (9.83)$$

$$[ \mathbf{A}_{k,2}^{(e)} ] \{ \mathbf{u}_{k,2}^{(e)} \} - [ \mathbf{A}_{k,1}^{(e)} ] \{ \mathbf{u}_{k,1}^{(e)} \} - \{ \mathbf{f}^{(e)} \} = 0 \quad (9.84)$$

The common equations system of contacting bodies, which is formed from the equations systems for finite elements (9.82) - (9.84), accept the following form:

$$\begin{array}{ccccc}
 K_{11}^{(1)} & K_{12}^{(1)} & 0 & 0 & 0 \\
 K_{21}^{(1)} & K_{22}^{(1)} & 0 & 0 & -A_{k,1}^T \\
 0 & 0 & K_{11}^{(2)} & K_{12}^{(2)} & 0 \\
 0 & 0 & K_{21}^{(2)} & K_{22}^{(2)} & A_{k,2}^T \\
 0 & -A_{k,1}^T & 0 & A_{k,2}^T & 0
 \end{array}
 \begin{array}{l}
 \left\{ \begin{array}{l} U_1 \\ U_{1,k} \\ U_2 \\ U_{2,k} \\ q \end{array} \right\} = \left\{ \begin{array}{l} P_1 \\ P_{1,k} \\ P_2 \\ P_{2,k} \\ F \end{array} \right\}
 \end{array}
 \quad (9.85)$$

If the *initial gap*  $\Delta_k$  is zero and  $A_{k,1} = A_{k,2}$ , then the vector of contact efforts  $\{q\}$  can be eliminated from the equations system (9.85), what reduces the equations system to the following form:

$$\begin{array}{ccc}
 K_{11}^{(1)} & K_{12}^{(1)} & 0 \\
 K_{21}^{(1)} & K_{22}^{(1)} + K_{11}^{(2)} & K_{12}^{(2)} \\
 0 & K_{21}^{(2)} & K_{22}^{(2)}
 \end{array}
 \begin{array}{l}
 \left\{ \begin{array}{l} U_1 \\ U_k \\ U_2 \end{array} \right\} = \left\{ \begin{array}{l} P_1 \\ P_{1,k} + P_{2,k} \\ P_2 \end{array} \right\} ,
 \end{array}
 \quad (9.86)$$

or in the matrix form  $[K]\{U\} = \{P\}$ . (9.87)

By introducing the boundary conditions into the common system, and by solving this system, we obtain a *vector of displacements*  $\{U\}$ . By substituting a part of the vector  $\{U\}$  belonging to the first body, into the equations system of the first body, we obtain the *sought vector of contact efforts*  $\{q\}$ .

### 9.4.2. Input data for the solution of the problem

For the solution of a problem of *contact of elastic bodies* the **Contact** program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoin*), number of groups of finite elements (*ngr*) and the number of pairs of finite elements which are in the contact (*nelemk*) and the number of the known boundary conditions (*nbond*) are coded. Into the corresponding group of the finite elements are included those elements, which have identical values of an *elastic modulus*, *area* and the *Poisson coefficient*.

In the *second* line, the number of nodes, in which the additional stiffnesses (*nst*) are known, number of nodes, to which the external forces (*nforc*) are applied, number of the sides of finite elements, onto which the distributed loadings (*nq*) are acted, and a code of the plane tense state (*kstr*), are recorded; *kstr=0* for the *plane stress state*, and *kstr=1* for the *plane strain*.

In the *third* line, a print code of intermediate results (*kprint*), a problem type (*ngama*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, a precision of the solution (*toler*) and density of a material (*tankis*) are coded. If *kprint=0*, then the intermediate results are not printed; otherwise, they are printed out. If the problem is solved in the *cartesian* frame, then *ngama=0*, if *ngama=1* – that in the *cylindrical* frame. If parameter *ksolve=0*, then for the solution the *'solve'* function of the *Maple-language* is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this

element {array **Mtop**(*nelem*, *nmode*), where *nmode* – number of nodes of the finite element, i.e. *nmode*=4} are coded, and also the number of group, to which belongs this finite element {array **Mgr**(*nelem*)}, and the number of a body, to which belongs the given finite element {array **Mkel**(*nelem*)}. After of arrays **Mtop**, **Mgr** and **Mkel** the elements of **Mkont**(*nelemk*, 4) array are coded line by line. Into each line of the file the line number and elements of **Mkont** array is recorded. Into each line of **Mkont** array the numbers of contacting elements of the first and the second bodies, and also two nodes of finite elements, being in contact, are coded.

After of array **Mkont** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and an element of **Lbond** array is recorded. Into each line of **Lbond** array the number of a node, in which the boundary condition is given, is recorded. After of array **Lbond** the elements of **Lst**(*nst*) array are coded line by line. Into each line of the file the line number and elements of **Lst** array are recorded. Into each line of **Lst** array the number of a node, in which the additional stiffness is presetted, are recorded.

After of array **Lst** the elements of **Lforc**(*nforc*) array are coded line by line. Into each line of the file a line number and the element of **Lforc** array are recorded. Into each line of **Lforc** array the number of a node, in which the external forces operate, are recorded. After of array **Lforc** the elements of **Lq**(*nq*, 3) array are coded line by line. Into each line of the file a line number and the elements of **Lq** array are recorded. Into each line of **Lq** array the number of a finite element and two numbers of nodes, being on a side of a finite element, onto which the distributed loading affects, is recorded.

Behind of array **Lq** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, while into the *second* – the *y*-coordinates of nodes of finite elements are recorded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded. Behind of array **Coord** the elements of **Bond**(*nbond*) array are recorded line by line. Into each line of the datafile a line number of **Bond** array and a value of boundary condition are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

Behind of **Bond** array the elements of **Pgr**(*ngr*, 2) array are coded line by line. Into each line of the datafile a line number of **Pgr** array, values of an elastic modulus and the Poisson coefficient of the corresponding group are recorded. The line number of **Pgr** array should strictly correspond to the number of group of finite elements. After of array **Pgr** the elements of **Storis**(*nelem*, *nmode*) array are recorded line by line. Into each line of the datafile, a line number of **Storis** array and values of thickness of an element in each node, are recorded. In case of the *axial-symmetric* problem (*ngama*=1) thickness of elements is not considered and the given array is not formed.

After of array **Storis** the elements of **Ast**(*nst*, 2) array are recorded line by line. Into each line of the datafile, a line number of **Ast** array and the values of the stiffnesses corresponding to each degree of freedom in the given node, are coded. The lines of **Ast** array should strictly correspond to the lines of **Lst** array. After of array **Ast** the elements of **Forc**(*nforc*, 2) array are recorded line by line. Into each line, a line number of **Forc** array and the values of external forces acting in a direction of each degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array.

After of array **Forc** the elements of **Fq**(*nq*, 2) arrays are recorded line by line. Into each line, a line number of array **Fq** and the values of intensities of distributed loadings acting onto a side of a finite element in its two nodes, are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*

*nelem, npoin, ngr, nelemk, nbond, nst, nforc, nq, kstr, kprint, ngama, ksolve, niter, toler, tankis*

Text line \*

Arrays *Mtop(nelem, nnode), Mgr(nelem), Mkel(nelem)*

Text line \*

Array *Mkont(nelemk, 4)*

Text line \*

Array *Lbond(nbond)*

Text line \*

Array *Lst(nst)*

Text line \*

Array *Lforc(nforc)*

Text line \*

Array *Lq(nq, 3)*

Text line \*

Array *Coord(npoin, ndime)*

Text line \*

Array *Bond(nbond)*

Text line \*

Array *Pgr(ngr, 2)*

Text line \*

Array *Storis(nelem, nnode)*

Text line \*

Array *Ast(nst, 2)*

Text line \*

Array *Forc(nforc, 2)*

Text line \*

Array *Fq(nq, 2)*

### 9.4.3. Brief description of the Contact program solving the problem

The *Contact* program was programmed on the *Maple*-language; it consists of the basic program and 50 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of *displacements* and *stresses* in nodes of finite elements, and also the *contact efforts* and *reactions in bearings*. The calculation results are output on the monitor and are recorded into a datafile, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of *displacements* and *stresses* in nodes of finite elements, and also the *contact efforts* and *reactions in bearings* are recorded.

### 9.4.4. An example of use of the Maple-program Contact

As an example of application of the *Contact* program, an *axial-symmetric problem of contact of two elastic bodies* (fig. 9.11) is considered. The first body is made from *aluminium bronze*, the second – from *carbon steel*; in addition onto upper part of the second body the given distributed loading affects.

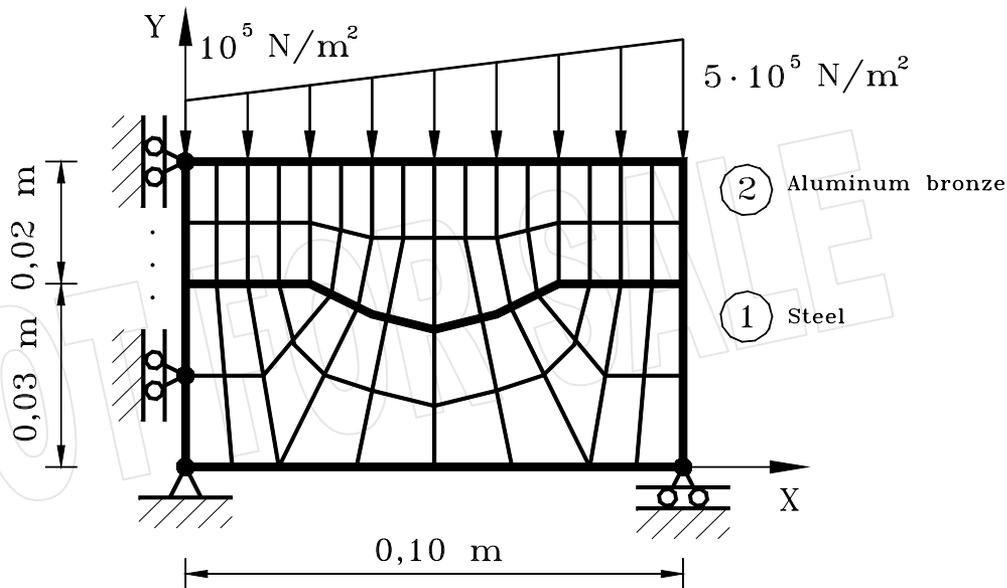


Fig. 9.11. The calculated scheme of two contacting bodies

The elastic modulus and the Poisson coefficient of contacting bodies are used: carbon steel -  $E=200$  GPa;  $\mu=0.25$  and aluminium bronze -  $E=103$  GPa;  $\mu=0.25$ .

Input data for the test example: number of the finite elements  $nelem=56$ , number of nodes  $npoin=73$ , number of groups of finite elements  $ngr=2$ , number of boundary conditions  $nbond=7$ ,  $kstr=0$ ,  $kprint=0$ ,  $nq=16$ ,  $nst=1$ ,  $nforc=1$ ,  $ksolve=1$ ,  $niter=500$ ,  $toler=10^{-6}$ ,  $ngama=1$ ,  $tankis=7850$  kg/m<sup>3</sup>, number of finite elements  $nelemk=16$ , being in the contact.

Results of calculation over the test example:

Values of displacements ( $ux, uy$ ) and equivalent stresses ( $sekv$ ) in nodes ( $node$ ) of finite elements calculated by means of the *Contact*-program:

node=1	ux=0e-01	uy=0e-01	sekv=6.872687e+05
node=2	ux=-1.889190e-08	uy=-5.589225e-07	sekv=8.544628e+05
node=3	ux=-1.366491e-08	uy=-6.393730e-07	sekv=3.336746e+05
node=4	ux=0e-01	uy=-5.284322e-07	sekv=2.523887e+06
node=5	ux=7.684968e-09	uy=-5.183952e-07	sekv=4.432525e+06
node=6	ux=1.071429e-08	uy=-6.264503e-07	sekv=4.090047e+06
node=7	ux=0e-01	uy=-4.026677e-09	sekv=2.186861e+06
node=8	ux=-5.464027e-07	uy=9.669949e-09	sekv=4.733821e+06
node=9	ux=-6.179174e-07	uy=1.815237e-08	sekv=5.560037e+06
node=10	ux=2.111593e-08	uy=-6.879133e-07	sekv=5.699010e+06
node=11	ux=-6.249470e-07	uy=3.310639e-07	sekv=7.484986e+06
node=12	ux=1.485548e-08	uy=-6.798275e-07	sekv=7.015926e+06
node=13	ux=-4.699435e-07	uy=5.206621e-07	sekv=7.854871e+06
node=14	ux=-6.690751e-07	uy=1.524689e-08	sekv=6.237670e+06
node=15	ux=-6.446883e-07	uy=2.658586e-07	sekv=5.103638e+06
node=16	ux=2.755204e-08	uy=-6.895088e-07	sekv=1.789983e+05
node=17	ux=9.402307e-08	uy=-6.786477e-07	sekv=2.047884e+05
node=18	ux=3.703986e-08	uy=-7.032617e-07	sekv=5.338862e+06
node=19	ux=5.903406e-08	uy=-6.886220e-07	sekv=4.871322e+06
node=20	ux=-6.913254e-07	uy=1.517740e-07	sekv=6.674297e+06

node=21	ux=-6.928302e-07	uy=2.060542e-08	sekv=5.995964e+06
node=22	ux=1.650781e-07	uy=-5.960480e-07	sekv=2.035381e+05
node=23	ux=2.284818e-07	uy=-4.620497e-07	sekv=2.250533e+05
node=24	ux=7.320404e-08	uy=-6.401693e-07	sekv=4.077963e+06
node=25	ux=7.613493e-08	uy=-5.676183e-07	sekv=3.287177e+06
node=26	ux=-6.524434e-07	uy=-1.114272e-07	sekv=5.038572e+06
node=27	ux=-5.620416e-07	uy=-2.784829e-07	sekv=4.074448e+06
node=28	ux=-3.902766e-07	uy=-4.488060e-07	sekv=4.144949e+06
node=29	ux=1.105259e-08	uy=-5.327301e-07	sekv=3.669347e+06
node=30	ux=6.315069e-08	uy=-4.356695e-07	sekv=1.732940e+06
node=31	ux=-5.183901e-07	uy=-2.567259e-07	sekv=3.039137e+06
node=32	ux=-5.060806e-07	uy=-5.480800e-08	sekv=3.584920e+06
node=33	ux=-4.218640e-07	uy=-6.091178e-08	sekv=2.850633e+06
node=34	ux=2.779923e-07	uy=-3.145566e-07	sekv=3.527406e+05
node=35	ux=3.303709e-07	uy=0e-01	sekv=2.726060e+05
node=36	ux=5.385568e-08	uy=-3.114623e-07	sekv=1.459848e+06
node=37	ux=6.174837e-08	uy=-1.976635e-07	sekv=7.385982e+05
node=38	ux=-3.437210e-07	uy=-5.504054e-08	sekv=2.161546e+06
node=39	ux=-2.888242e-07	uy=-4.307447e-08	sekv=9.498025e+05
node=40	ux=0e-01	uy=-5.123126e-07	sekv=4.667276e+05
node=41	ux=2.727132e-08	uy=-5.562061e-07	sekv=1.365787e+06
node=42	ux=9.515310e-09	uy=-6.337510e-07	sekv=1.867586e+06
node=43	ux=0e-01	uy=-5.721101e-07	sekv=1.164844e+05
node=44	ux=2.069160e-08	uy=-5.929236e-07	sekv=2.100278e+05
node=45	ux=3.319729e-08	uy=-6.395671e-07	sekv=1.447777e+05
node=46	ux=-2.585264e-10	uy=-6.793177e-07	sekv=2.230418e+06
node=47	ux=-1.116607e-08	uy=-7.090815e-07	sekv=3.172500e+06
node=48	ux=1.986370e-08	uy=-6.878607e-07	sekv=8.747554e+04
node=49	ux=-2.839046e-09	uy=-7.190123e-07	sekv=6.780511e+04
node=50	ux=-1.350442e-08	uy=-7.150942e-07	sekv=3.081495e+06
node=51	ux=-1.698134e-08	uy=-7.196213e-07	sekv=2.007646e+06
node=52	ux=-2.242193e-08	uy=-7.308900e-07	sekv=6.114055e+04
node=53	ux=-4.255507e-08	uy=-7.350116e-07	sekv=5.519026e+04
node=54	ux=-2.598259e-08	uy=-7.191631e-07	sekv=1.559845e+06
node=55	ux=-3.798129e-08	uy=-7.060526e-07	sekv=1.365001e+06
node=56	ux=-6.335586e-08	uy=-7.325587e-07	sekv=4.541293e+04
node=57	ux=-8.473615e-08	uy=-7.231643e-07	sekv=3.557041e+04
node=58	ux=-4.895389e-08	uy=-6.794545e-07	sekv=1.195766e+06
node=59	ux=-5.845115e-08	uy=-6.472910e-07	sekv=9.659096e+05
node=60	ux=-1.054324e-07	uy=-7.064001e-07	sekv=3.161979e+04
node=61	ux=-1.242529e-07	uy=-6.827770e-07	sekv=3.372002e+04
node=62	ux=-7.700787e-08	uy=-6.182935e-07	sekv=1.524948e+06
node=63	ux=-9.505180e-08	uy=-5.954382e-07	sekv=1.901057e+06
node=64	ux=-1.405974e-07	uy=-6.536895e-07	sekv=3.765517e+04
node=65	ux=-1.545643e-07	uy=-6.191520e-07	sekv=4.811689e+04
node=66	ux=-1.028267e-07	uy=-5.294662e-07	sekv=1.505903e+06
node=67	ux=-1.043443e-07	uy=-4.601612e-07	sekv=1.344740e+06
node=68	ux=-1.661336e-07	uy=-5.591956e-07	sekv=7.084353e+04
node=69	ux=-1.667175e-07	uy=-4.981396e-07	sekv=9.888658e+04
node=70	ux=-9.890075e-08	uy=-3.972580e-07	sekv=1.174050e+06
node=71	ux=-8.728983e-08	uy=-3.492912e-07	sekv=5.470451e+05
node=72	ux=-1.581183e-07	uy=-4.431145e-07	sekv=1.196108e+05
node=73	ux=-1.469398e-07	uy=-3.951914e-07	sekv=6.323479e+04

In addition, the allocation of *normal* and *tangential* contact efforts in the direction of coordinate axis X is represented on the *fig. 9.12*.

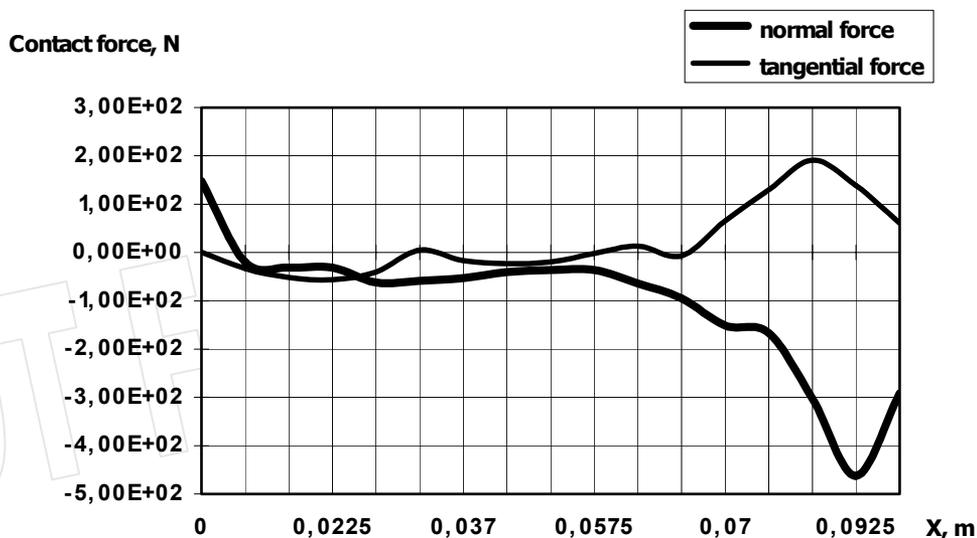


Fig. 9.12. Allocation of contact efforts in the direction of axis X

The *Contact* program is intended for the solution of the *contact problems* of the elasticity theory. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles *Mb6\_4\_new.mws*, *Mb6\_4.dat* and *Mb6\_4\_1.rez* accordingly.

# Chapter 10.

## Dynamic problems of the theory of elasticity

*Dynamic problems* of the theory of elasticity are considerably interesting when designing many objects of mechanical engineering. This interest becomes apparent in case of necessity of definition of *free* and *forced oscillations* of elastic bodies. *Own* or *free* oscillations are motions of such oscillatory system, which are not exposed to any external action after a short-term perturbation, i.e. when moving, outside energy is not fed. When investigating oscillations of elastic systems, the *least frequencies* and *forms of eigentones* are of a special interest because the oscillation amplitude is *peak* at the least frequencies. At forced oscillations, the system is exposed to external actions, which define the period of oscillations. When disturbing frequency coincides with the free-running frequency of the system, *resonance* occurs. The oscillation amplitude will increase on resonant frequencies. If oscillation amplitudes of an elastic body are known, *tense state* of an explored body can be defined. The behaviour of elastic system at *transient loadings* is of great interest. The problems, considered in the given chapter, represent practical interest for investigation of *free* and *forced* oscillations of elastic bodies of various nature and assignment.

### 10.1. The dynamic problem of beam systems

Composite beam constructions can be subjected to dynamic loads of various kinds. The action of external dynamic loads can vary both in *amplitude* and *frequency*. The solution of equations of motion of beam systems allows to define an *amplitude* and an *oscillation frequency* of their *constituents*. If amplitudes are known, it is possible to define *internal efforts*, *strains*, *stresses* etc., i.e. it is possible to define the tense system state in time as a whole. The problem of *free* and *forced* oscillations of the *linear beam system* is considered in the given section. The given problem is solved by means of the FEM.

#### 10.1.1. The calculated equations of beam system's dynamics

The following three groups of forces affect onto the construction at *dynamic loads*: *elasticity*, *inertia* and *resistance*, which, according to the principle of d'Alembert, can be changed by their *static equivalent* being defined by the following relation:  $\rho \frac{d^2 \mathbf{u}}{dt^2}$ , where:  $\rho$  – density of material and  $\mathbf{u}$  – displacement. The forces of resistance movement arise in places of friction between separate parts of the construction, from air resistance, etc. In general, these forces depend on velocity non-linearly. To simplify the enunciating, only linear resistance of viscous type is considered, which is statically equivalent to the force being defined by the following relation:  $\xi \frac{d\mathbf{u}}{dt}$ , where:  $\xi$  – coefficient characterizing *dissipative properties* of the construction.

Therefore, *nodal forces* of a finite element are formed of nodal forces conditioned by *elastic strain* of the construction, *inertial* and *dissipative* forces. *Movement equations* of a *beam finite element* can be

derived by several methods (Newton's law, equations of Lagrange, principle of Hamilton etc.), which as a result reduce to the following expressions:

$$[\mathbf{m}^{(e)}]\{\ddot{\mathbf{u}}^{(e)}\} + [\mathbf{c}^{(e)}]\{\dot{\mathbf{u}}^{(e)}\} + [\mathbf{k}^{(e)}]\{\mathbf{u}^{(e)}\} = \{\mathbf{f}^{(e)}(\mathbf{t})\} \quad (10.1)$$

where  $[\mathbf{m}^{(e)}]$ ,  $[\mathbf{c}^{(e)}]$ ,  $[\mathbf{k}^{(e)}]$  – matrices of *masses*, *damping* and *stiffness*;  $\{\ddot{\mathbf{u}}^{(e)}\}$ ,  $\{\dot{\mathbf{u}}^{(e)}\}$ ,  $\{\mathbf{u}^{(e)}\}$  – vectors of *accelerations*, *velocities* and *displacements* of nodes of finite elements;  $\{\mathbf{f}^{(e)}(\mathbf{t})\}$  – vector of *external loadings*, which consists of *surface* and *concentrated forces*. The *stiffness matrix* of a finite element is defined from expression (9.30), while *matrix of masses* of a beam finite element is defined as follows:

$$[\mathbf{m}^{(e)}] = \int_0^L \rho [\mathbf{N}]^T \mathbf{A}(\mathbf{x}) [\mathbf{N}] \, d\mathbf{x}, \quad (10.2)$$

where  $[\mathbf{N}]$  – matrix of shape functions {see formulas (9.25) and (9.27)}. The *matrix of damping* of a beam finite element is defined by the following relation:

$$[\mathbf{c}^{(e)}] = \int_0^L \xi [\mathbf{N}]^T \mathbf{A}(\mathbf{x}) [\mathbf{N}] \, d\mathbf{x} \quad (10.3)$$

If the coefficient of damping  $\xi$  is unknown (*in most cases it is, indeed*), some authors suggest to define a matrix of the damping as follows [85]:

$$[\mathbf{c}^{(e)}] = \alpha_1 [\mathbf{k}^{(e)}] + \alpha_2 [\mathbf{m}^{(e)}] \quad (10.4)$$

where:  $\alpha_1$ ,  $\alpha_2$  – coefficients being defined by two pairs of coefficients of *damping* and *frequencies* corresponding to them, namely:

$$\alpha_1 = \frac{2(\xi_2 \omega_2 - \xi_1 \omega_1)}{\omega_2^2 - \omega_1^2}; \quad \alpha_2 = \frac{2\omega_1 \omega_2 (\xi_1 \omega_2 - \xi_2 \omega_1)}{\omega_2^2 - \omega_1^2} \quad (10.5)$$

Thus, the common system of dynamic balance is formed of equations systems of finite elements (10.1):

$$[\mathbf{M}]\{\ddot{\mathbf{U}}\} + [\mathbf{C}]\{\dot{\mathbf{U}}\} + [\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{F}(\mathbf{t})\} \quad (10.6)$$

To solve the given equations system, *initial conditions* shall be known, namely:

$$\{\mathbf{u}(\mathbf{x}, \mathbf{t} = 0)\} = \{\mathbf{u}_0(\mathbf{x})\}; \quad \{\dot{\mathbf{u}}(\mathbf{x}, \mathbf{t} = 0)\} = \{\dot{\mathbf{u}}_0(\mathbf{x})\} \quad (10.7)$$

and also to define *boundary conditions*:

$$\{\mathbf{u}(\mathbf{t}, \mathbf{x} = \mathbf{x}_0)\} = \{\mathbf{u}_0(\mathbf{t})\} \quad (10.8)$$

The obtained dynamical equations system is integrated by the Newmark method (*method of the generalized acceleration*). The *vectors of velocities* and *displacements* are approximated by the following relations:

$$\{\dot{\mathbf{U}}_{t+\tau}\} = \{\dot{\mathbf{U}}_t\} + [(1-\delta)\ddot{\mathbf{U}}_t + \delta\ddot{\mathbf{U}}_{t+\tau}]\tau; \quad \{\mathbf{U}_{t+\tau}\} = \{\mathbf{U}_t\} + \{\dot{\mathbf{U}}_t\}\tau + \left[\left(\frac{1}{2} - \alpha\right)\ddot{\mathbf{U}}_t + \alpha\ddot{\mathbf{U}}_{t+\tau}\right]\tau^2 \quad (10.9)$$

where  $\delta, \alpha$  – parameters defining precision and stability of integration ( $\delta = \frac{1}{2}, \alpha = \frac{1}{6} \dots \frac{1}{4}$ ). After substitution of the obtained expressions into (10.9), the following equation defines the *vector of displacements*:

$$[\mathbf{K}]\{\mathbf{U}_{t+\tau}\} = \{\mathbf{P}_{t+\tau}\}; \quad (10.10)$$

where:  $[\mathbf{K}] = [\mathbf{K}] + \frac{1}{\alpha\tau^2}[\mathbf{M}] + \frac{\delta}{\alpha\tau}[\mathbf{C}];$

$$\{\mathbf{P}_{t+\tau}\} = \{\mathbf{P}_{t+\tau}\} + [\mathbf{M}]\left(\frac{1}{\alpha\tau^2}\{\mathbf{U}_t\} + \frac{1}{\alpha\tau}\{\dot{\mathbf{U}}_t\} + \left(\frac{1}{2\alpha} - 1\right)\{\ddot{\mathbf{U}}_t\}\right) + [\mathbf{C}]\left(\frac{\delta}{\alpha\tau}\{\mathbf{U}_t\} + \left(\frac{\delta}{\alpha} - 1\right)\{\dot{\mathbf{U}}_t\} + \frac{\tau}{2}\left(\frac{\delta}{\alpha} - 2\right)\{\ddot{\mathbf{U}}_t\}\right) \quad (10.11)$$

where  $\tau$  – integration step in time. Vectors of *accelerations* and *velocities* at moment  $t+\tau$  are defined as follows:

$$\{\ddot{\mathbf{U}}_{t+\tau}\} = \frac{1}{\alpha\tau^2}(\{\mathbf{U}_{t+\tau}\} - \{\mathbf{U}_t\}) - \frac{1}{\alpha\tau}\{\dot{\mathbf{U}}_t\} - \left(\frac{1}{2\alpha} - 1\right)\{\ddot{\mathbf{U}}_t\}; \quad \{\dot{\mathbf{U}}_{t+\tau}\} = \{\dot{\mathbf{U}}_t\} + \tau(1 - \delta)\{\ddot{\mathbf{U}}_t\} + \delta\tau\{\ddot{\mathbf{U}}_{t+\tau}\} \quad (10.12)$$

The definition of *natural frequencies* and *forms of free oscillations* of a beam system is generally reduced to the following solution of the problem of *eigenvalues*:

$$(-\omega^2[\mathbf{M}] + i\omega[\mathbf{C}] + [\mathbf{K}])\{\mathbf{U}\} = \mathbf{0} \quad (10.13)$$

where  $\omega$  – natural frequency. To evaluate  $n$  of the *least eigenvalues* and *eigenvectors* for the system  $[\mathbf{K}]\{\mathbf{U}\} = \lambda[\mathbf{M}]\{\mathbf{U}\}$ , the method of iterations in subspace is used [85]. The efficiency of the method is defined by the factor that the total number of iterations up to achievement of required precision depends not on the proximity of iterated vectors to the eigenvectors but on the proximity of  $\mathbf{q}$ -dimensional space of iterated vectors to the subspace of the eigenvectors. The sought *minimum eigenvalues* and the vectors should satisfy the following relation:

$$[\mathbf{K}][\Phi] = [\mathbf{M}][\Phi][\lambda] \quad (10.14)$$

where:  $[\Phi] = [\{\mathbf{U}\}_1 \quad \{\mathbf{U}\}_2 \quad \dots \quad \{\mathbf{U}\}_n]$  – a matrix of *eigenvectors*;  $[\lambda] = \text{diag}(\lambda_i), i=1 \dots n$  – a diagonal matrix of *eigenvalues*. The eigenvectors should obey orthogonalities of the following form:

$$[\Phi]^T [\mathbf{M}][\Phi] = [\mathbf{E}]; \quad (10.15)$$

$$[\Phi]^T [\mathbf{K}][\Phi] = [\lambda] \quad (10.16)$$

where  $[\mathbf{E}]$  – *unit matrix*. On the initial stage, it is necessary to generate a matrix of *initial approximation* of the eigenvectors  $[\Phi]_{i=0}$  of dimensionality ( $\mathbf{N} \times \mathbf{n}$ ), where  $\mathbf{N}$  – total number of the equations. The iterative procedure of this method in each  $i$ -iteration includes the following *basic* stages:

- orthogonalization of a matrix  $[\Phi]_{i-1}$ ;
- forming of a matrix  $[\mathbf{R}]_i = [\mathbf{M}][\Phi]_{i-1}$ ;
- solution of a system  $[\mathbf{K}][\Phi]_i = [\mathbf{R}]_i$ ;

- evaluation of matrices  $[\mathbf{K}]_i = [\Phi]_i^T [\mathbf{R}]_i$  and  $[\mathbf{M}]_i = [\Phi]_i^T [\mathbf{M}] [\Phi]_i$ ;
- solution of eigenvalues problem of  $n$ -th order  $[\mathbf{K}]_i [\mathbf{Q}]_i = [\mathbf{M}]_i [\mathbf{Q}]_i [\Lambda]_i$ ;
- forming a new basis  $[\Phi]_i = [\Phi]_i [\mathbf{Q}]$ ;
- checkout of convergence  $\frac{|\lambda_{k,i} - \lambda_{k,i-1}|}{\lambda_{k,i}} \leq \varepsilon$  for  $k=1..n$

Generally, at the choice  $q = \min(2*n, n+8)$  and rational choice of an initial matrix  $[\Phi]_{i=0}$  the algorithm of iterations in subspace ensures obtaining of *natural numbers* with the required precision.

### 10.1.2. Input data for the solution of the problem

To solve the *dynamic problem of beam constructions*, the *Dynamic\_strypas* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), number of groups of finite elements (*ngr*) and the number of known boundary conditions (*nbond*) are coded. Those elements into the corresponding group of finite elements, which have: (1) identical values of an *elastic modulus*, (2) density of a material, (3) coefficients  $\alpha_1$  and  $\alpha_2$ , which define matrix of damping, and (4) coefficients  $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2$ , which define a cross-sectional area ( $\mathbf{A}(x) = \mathbf{A}_0 + \mathbf{A}_1 x + \mathbf{A}_2 x^2$ ), and also (5) coefficients  $I_0, I_1, I_2, I_3, I_4$ , which define a moment of inertia of cross-section ( $I(x) = I_0 + I_1 x + I_2 x^2 + I_3 x^3 + I_4 x^4$ ).

In the *second* line, the number of finite elements, for which the coefficients of the elastic basis are known (*nkd*); the number of nodes, in which additional coefficients of stiffness and damping are given (*nst*); the number of nodes, in which external forces affect (*nforc*); and the number of finite elements, onto which the distributed loadings affect (*nq*), are recorded.

In the *third* line, the print code of intermediate results (*kprint*), problem type (*ngama*), the code of the system solution of algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equation system, and precision of the solution (*toler*) are written. If *kprint*=0, then intermediate results are not printed; otherwise, they are printed out. if parameter *ksolve*=0, then to solve, the *'solve'* function of the *Maple*-language is used; otherwise, the equation system is solved by the method of *conjugate gradients*.

In the *fourth* line, the number of integration steps (*ntime*), and integration step (*dtime*) are coded. In the *fifth* line, the number of eigenvalues (*neigen*); the number of iterations (*nitero*); precision of definition of eigenvalues and vectors (*tolero*), are coded. If definitions of eigenvalues and vectors are not required, then relation *neigen*=0 is supposed. In the *sixth* line, the number of degrees of freedom, for which into the file the values of results are recorded (*nout*); and the number indicating through how much of integration steps the results of evaluations are recorded (*nstep*), are coded.

In each subsequent *nelem* line, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode* = 2} are coded, and also the number of group, to which this finite element {array **Mgr**(*nelem*)} belongs. After arrays **Mtop** and **Mgr**, the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into each line of **Lbond** array the numbers of nodes, where boundary conditions are known, are coded.

After array **Lbond**, the elements of **Lkd**(*nkd*) array are coded line by line. Into each line of the file

the line number and an element of **Lkd** array are recorded. Into each line of **Lkd** array, the numbers of finite elements which lie on the elastic basis, are coded. After array **Lkd**, the elements of **Lst(nst)** array are coded line by line. Into each line of the file the line number and an element of **Lst** array are recorded. Into each line of **Lst** array the number of a node, in which additional coefficients of stiffness and damping are given, are recorded.

After array **Lst**, the elements of **Lforc(nforc)** array are coded line by line. Into each line of the file, a line number and the element of **Lforc** array are recorded. Into each line of **Lforc** array the numbers of nodes of finite elements, where external forces affect, are recorded. After array **Lforc**, the elements of **Lq(nq)** array are coded line by line. Into each line of the file, a line number and the element of **Lq** array are recorded. Into each line of **Lq** array the number of a finite element, onto which the distributed loading affects, is recorded.

After array **Lq**, the elements of **Lout(nout)** array are coded line by line. Into each line of the file, a line number and the element of **Lout** array are recorded. Into **Lout** array, the numbers of degrees of freedom, for which the values of results are recorded into the file, are coded. Behind array **Lout** file the elements of **Coord(npoin, ndime)** array, where *ndime* – dimensionality of the problem (*ndime* = 2), are recorded into the data line by line. Into the *first* column of the **Coord** array the *x*-coordinates, while into the *second* – the *y*-coordinates of nodes of finite elements are recorded. Into each line of the file the number of a node as well as its (*x*, *y*)-coordinates are recorded. Behind array **Coord** the elements of **Bond(nbond)** array are recorded line by line. Into each line of the datafile a line number of **Bond** array and a value of boundary condition are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

Behind **Bond** array, the elements of **Pgr(ngr, 12)** array are coded line by line. Into each line of the datafile a line number of **Pgr** array, values of an elastic modulus, density of material, coefficients  $\alpha_1$  and  $\alpha_2$ , which define the matrix of damping as well as coefficients **A<sub>0</sub>**, **A<sub>1</sub>**, **A<sub>2</sub>** and **I<sub>0</sub>**, **I<sub>1</sub>**, **I<sub>2</sub>**, **I<sub>3</sub>**, **I<sub>4</sub>**, are recorded. The line number of **Pgr** array should strictly corresponds to the number of a group of finite elements. After array **Pgr**, the elements of **Akd(nkd, 2)** array are recorded line by line. Into each line of the datafile, line number of **Akd** array and two values of coefficients of elastic basis in each node of a finite element, are recorded. The lines of **Akd** array should correspond to the lines of **Lkd** array.

After array **Akd**, the elements of **Ast(nst, 6)** array are recorded line by line. Into each line of the datafile, line number of **Ast** array, three values of additional coefficients of stiffness, and three values of additional coefficients of damping, which correspond to each degree of freedom in a node, are recorded. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After array **Ast** the elements of **Forc(nforc, 12)** array are recorded line by line. Into each line, line number of **Forc** array, and three groups of coefficients **a<sub>i</sub>** (*i*=0..3), by which the concentrated forces  $\mathbf{f}_k(\mathbf{t}) = [\mathbf{a}_0 + \mathbf{a}_1 \sin(\mathbf{a}_2 \mathbf{t})] \mathbf{e}^{\mathbf{a}_3 \mathbf{t}}$  (*k*=1, 2, 3) are defined, which act in the direction of each degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array.

After array **Forc** the elements of **Fq(nq, 6)** arrays are recorded line by line. Into each line, line number of array **Fq** and the values of coefficients **q<sub>1</sub>**, **q<sub>2</sub>**, **b<sub>0</sub>**, **b<sub>1</sub>**, **b<sub>2</sub>**, **b<sub>3</sub>** by which the surface forces are defined, are computed as follows  $\mathbf{f}_s(\mathbf{x}, \mathbf{t}) = [\mathbf{q}_1 \mathbf{N}_1(\mathbf{x}) + \mathbf{q}_2 \mathbf{N}_4(\mathbf{x})][\mathbf{b}_0 + \mathbf{b}_1 \sin(\mathbf{b}_2 \mathbf{t})] \mathbf{e}^{\mathbf{b}_3 \mathbf{t}}$ , are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. When reading information from the datafile, these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nkd, nst, nforc, nq, kprint, ksolve, niter, toler, ntime, dtime, neigen, nitero, tolero, nout, nstep*  
Text line \*  
Arrays *Mtop(nelem, mmode), Mgr(nelem)*  
Text line \*  
Array *Lbond(nbond)*  
Text line \*  
Array *Lkd(nkd)*  
Text line \*  
Array *Lst(nst)*  
Text line \*  
Array *Lforc(nforc)*  
Text line \*  
Array *Lq(nq)*  
Text line \*  
Array *Coord(npoin, ndime)*  
Text line \*  
Array *Bond(nbond)*  
Text line \*  
Array *Pgr(ngr, 12)*  
Text line \*  
Array *Akd(nkd, 2)*  
Text line \*  
Array *Ast(nst, 6)*  
Text line \*  
Array *Forc(nforc, 12)*  
Text line \*  
Array *Fq(nq, 6)*

### 10.1.3. Brief description of the `Dynamic_strypas` program solving the problem

The `Dynamic_strypas` program was programmed on the `Maple`-language; it consists of the basic program and 35 procedures. All procedures can be divided into *three* groups: procedures for data entry, calculation and output of results. Memory size necessary for the solution of a concrete problem and time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of *displacements, velocities and accelerations* of nodes of finite elements in time dependence, *natural frequencies*, and *forms*. Calculation results are output on the monitor and are recorded into a file; therefore, a *qualifier* of a target file must be ascribed to variable `file_rez1` of the program. Into the file `file_rez1` values of *displacements, velocities and accelerations* of nodes of finite elements (*in time dependence*) in degrees of freedom indicated in the array `Lout(nout)`, *natural frequencies*, and *forms* are recorded.

### 10.1.4. An example of the use of `Maple`-program `Dynamic_strypas`

As an example of application of the program, the motion of the *beam construction* consisting of three various groups of beams is considered (*fig. 10.1*).

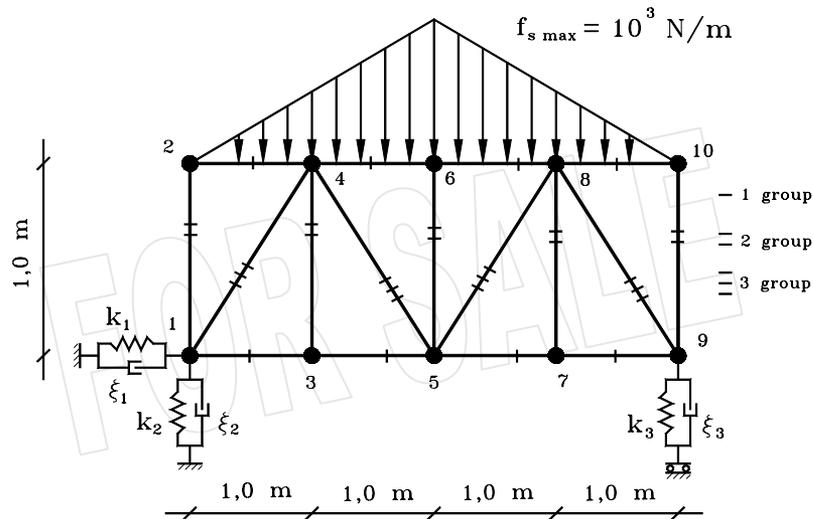


Fig. 10.1. The calculating circuit of beam construction

Short-term surface force  $f_s(x, t)$  of the following view affects onto the construction:

$$f_s(x, t) = q(x)[b_0 + b_1 \sin(b_2 t)]e^{b_3 t}, \text{ where } b_0 = 1 \frac{\text{N}}{\text{m}}, b_1 = b_2 = 0, b_3 = -700 \text{ s}^{-1}$$

Input data for the test example: number of finite elements  $nelem=17$ , number of nodes  $npoint=10$ , number of groups of finite elements  $ngr=3$ , number of boundary conditions  $nbond=1$ ,  $kprint=0$ ,  $nq=4$ ,  $nst=2$ ,  $nforc=1$ ,  $ksolve=0$ ,  $niter=500$ ,  $toler=10^{-9}$ ,  $ngama=1$ ,  $nkd=1$ ,  $ntime=500$ ,  $dtime=10^{-4} \text{ s}$ ,  $neigen=5$ ,  $niterv=100$ ,  $nout=2$ ,  $nstep=2$ ,  $tolerv=10^{-6}$ .

Parameters of the

first group:

$$E = 210 \text{ GPa}; \rho = 7850 \frac{\text{kg}}{\text{m}^3}; \alpha_1 = \alpha_2 = 0; A_0 = 2,5 \cdot 10^{-3} \text{ m}^2;$$

$$A_1 = A_2 = 0; I_0 = 6,25 \cdot 10^{-6} \text{ m}^4; I_1 = I_2 = I_3 = I_4 = 0$$

second group:

$$E = 205 \text{ GPa}; \rho = 7830 \frac{\text{kg}}{\text{m}^3}; \alpha_1 = \alpha_2 = 0; A_0 = 9 \cdot 10^{-4} \text{ m}^2;$$

$$A_1 = A_2 = 0; I_0 = 8,1 \cdot 10^{-7} \text{ m}^4; I_1 = I_2 = I_3 = I_4 = 0$$

third group:

$$E = 200 \text{ GPa}; \rho = 7800 \frac{\text{kg}}{\text{m}^3}; \alpha_1 = \alpha_2 = 0; A_0 = 4 \cdot 10^{-4} \text{ m}^2;$$

$$A_1 = A_2 = 0; I_0 = 1,6 \cdot 10^{-7} \text{ m}^4; I_1 = I_2 = I_3 = I_4 = 0$$

Coefficients of additional stiffness and damping:  $k_1 = k_2 = k_3 = 10^6 \frac{\text{N}}{\text{m}}; \xi_1 = \xi_2 = \xi_3 = 10^{-3} \frac{\text{N} \cdot \text{s}}{\text{m}}$

Results of calculation over the test example:

Natural frequencies (Hz):

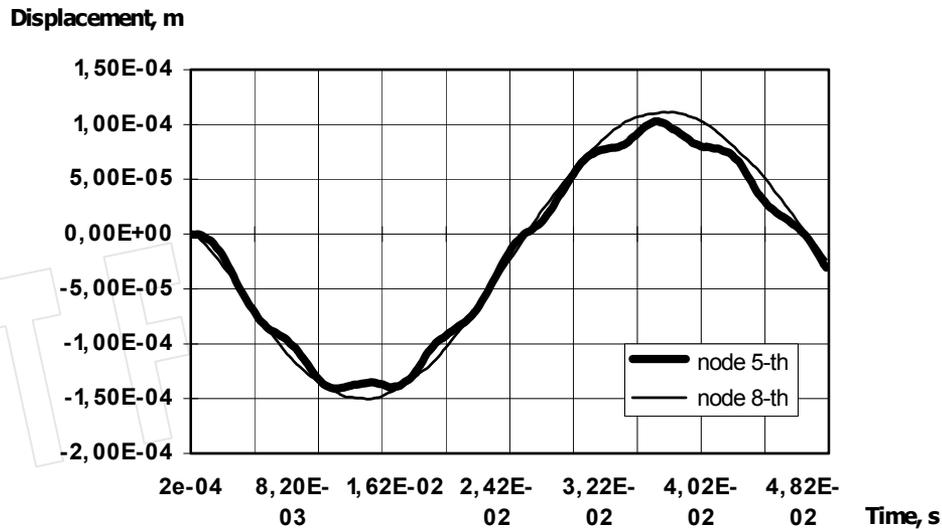
- 1  $w=1.056602e+02$
- 2  $w=2.406061e+02$

3 w=3.254788e+02  
 4 w=4.032010e+02  
 5 w=5.951930e+02

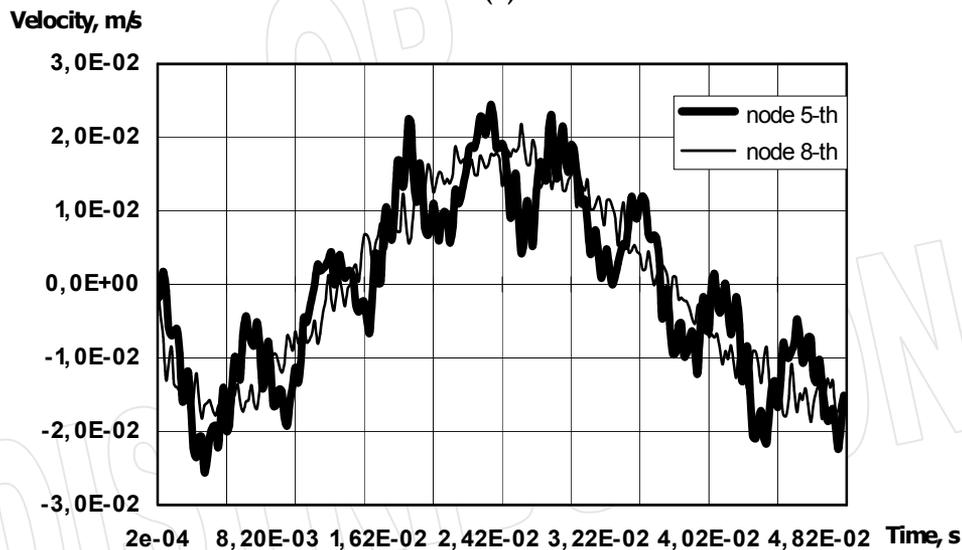
Natural forms:

1	-7.656174e-03	-6.620235e-03	3.439181e-02	3.102641e-02	3.923920e-03
2	-1.865802e-02	2.738153e-02	4.903726e-02	1.088007e-02	-1.571378e-01
3	-7.373096e-02	2.978907e-01	4.520510e-02	-2.619932e-01	-1.007419e+00
4	-7.647402e-03	-6.612651e-03	3.435241e-02	3.099086e-02	3.919424e-03
5	-8.047478e-03	-6.958593e-03	3.614957e-02	3.261216e-02	4.124470e-03
6	-2.120929e-04	-1.833951e-04	9.527295e-04	8.595004e-04	1.087012e-04
7	-1.310936e-02	-1.133556e-02	5.888773e-02	5.312529e-02	6.718772e-03
8	-6.734221e-02	1.708468e-01	-1.019606e-03	-1.055782e-01	5.665479e-02
9	-3.681489e-04	-3.183356e-04	1.653738e-03	1.491912e-03	1.886826e-04
10	-1.570576e-02	-1.358065e-02	7.055087e-02	6.364713e-02	8.049472e-03
11	-6.319233e-02	1.453212e-01	-1.120180e-02	-7.795672e-02	1.161505e-01
12	-5.084950e-04	-4.396918e-04	2.284178e-03	2.060660e-03	2.606124e-04
13	-1.570576e-02	-1.358065e-02	7.055087e-02	6.364713e-02	8.049472e-03
14	-8.022967e-02	6.223750e-03	-1.337995e-01	3.747631e-02	-2.801171e-02
15	-5.084950e-04	-4.396918e-04	2.284178e-03	2.060660e-03	2.606124e-04
16	-1.310936e-02	-1.133556e-02	5.888773e-02	5.312529e-02	6.718772e-03
17	-8.191523e-02	6.514043e-03	-1.509116e-01	6.745326e-02	-3.437774e-02
18	-3.681489e-04	-3.183356e-04	1.653738e-03	1.491912e-03	1.886826e-04
19	-1.310936e-02	-1.133556e-02	5.888773e-02	5.312529e-02	6.718772e-03
20	-6.750012e-02	-1.486113e-01	-9.602544e-03	-1.394112e-01	6.183119e-03
21	-3.681489e-04	-3.183356e-04	1.653738e-03	1.491912e-03	1.886826e-04
22	-1.570576e-02	-1.358065e-02	7.055087e-02	6.364713e-02	8.049472e-03
23	-6.372479e-02	-1.252466e-01	-1.810572e-02	-1.027069e-01	-2.330942e-02
24	-5.084950e-04	-4.396918e-04	2.284178e-03	2.060660e-03	2.606124e-04
25	-1.023503e-02	-8.850156e-03	4.597614e-02	4.147716e-02	5.245629e-03
26	-8.056250e-03	-6.966178e-03	3.618897e-02	3.264771e-02	4.128965e-03
27	8.620322e-02	2.673263e-01	-6.528493e-02	2.467533e-01	-2.615948e-01
28	-7.647402e-03	-6.612651e-03	3.435241e-02	3.099086e-02	3.919424e-03
29	-8.047478e-03	-6.958593e-03	3.614957e-02	3.261216e-02	4.124470e-03
30	-2.120929e-04	-1.833951e-04	9.527295e-04	8.595004e-04	1.087012e-04

In *fig. 10.2* dependences of vertical displacements (a) and velocities (b) of nodes with numbers 5 and 8 in time are represented.



(a)



(b)

Fig. 10.2. The dependences of vertical displacements (a) and velocities (b) of nodes with numbers 5 and 8 in time

The *Dynamic\_strypas* program is intended for dynamic calculation of the *beam systems* on the plane. Natural frequencies and forms are determined and the system of dynamic balance is integrated in time. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in files *Mb7\_1\_new.mws*, *Mb7\_1.dat* and *Mb7\_1\_1.rez* accordingly.

## 10.2. The plane problem of the theory of elasticity at dynamic loads

The *plane* problem of the theory of elasticity is the simplified *three-dimensional* problem of the theory of elasticity, however it is successfully applied to the solution of simple applied problems. This problem becomes complicated under the condition of activity of *dynamic loads*. *External loadings* can act with various frequencies and amplitudes; therefore, it is necessary to determine frequencies of

natural oscillations for elimination of *resonance oscillations* (if they are undesirable). The *plane problem* of the theory of elasticity is solved in *cartesian* or *cylindrical* axials, expanding the opportunities of the solution of a whole series of applied dynamic problems.

### 10.2.1. The calculated equations of the theory of elasticity problem at dynamic loads

The *plane problem* of the theory of elasticity under condition of *dynamic loads* is solved by the FEM. The linear two-dimensional four-nodal isoparametric finite element, considered above, is used (see fig. 8.2). The equation of motion of a finite element of an elastic body is defined by the *principle of virtual displacements* by the following relations:

$$\int_{V^{(e)}} (\delta\{\boldsymbol{\varepsilon}\}^T \{\boldsymbol{\sigma}\} + \delta\{\mathbf{u}\}^T \rho \{\ddot{\mathbf{u}}\} + \delta\{\mathbf{u}\}^T \xi \{\dot{\mathbf{u}}\}) dV = \int_{V^{(e)}} \delta\{\mathbf{u}\}^T \{\mathbf{f}_v\} dV + \int_{S^{(e)}} \delta\{\mathbf{u}\}^T \{\mathbf{f}_s\} dS + \sum_i \delta\{\mathbf{u}\}_i^T \{\mathbf{f}_k\}_i \quad (10.17)$$

where:  $\{\boldsymbol{\varepsilon}\}$ ,  $\{\boldsymbol{\sigma}\}$  – vectors of strains and stresses;  $\{\ddot{\mathbf{u}}\}$ ,  $\{\dot{\mathbf{u}}\}$ ,  $\{\mathbf{u}\}$  – vectors of accelerations, velocities and displacements;  $\rho$  – density of a material;  $\xi$  – coefficient describing dissipative properties of a material;  $\{\mathbf{f}_v\}$ ,  $\{\mathbf{f}_s\}$ ,  $\{\mathbf{f}_k\}$  – vectors of the volumetric, surface and concentrated forces accordingly. At the given suppositions the *displacements*, *velocities* and *acceleration* in a finite element are approximated by relations of the following form:

$$\{\mathbf{u}\} = [\mathbf{N}] \{\mathbf{u}^{(e)}\}; \quad \{\dot{\mathbf{u}}\} = [\mathbf{N}] \{\dot{\mathbf{u}}^{(e)}\}; \quad \{\ddot{\mathbf{u}}\} = [\mathbf{N}] \{\ddot{\mathbf{u}}^{(e)}\} \quad (10.18)$$

Hence, the *vector of strains* can be expressed via a vector of nodal displacements of a finite element  $\{\mathbf{u}^{(e)}\}$ , namely:

$$\{\boldsymbol{\varepsilon}\} = [\mathbf{B}] \{\mathbf{u}^{(e)}\} \quad (10.19)$$

where  $[\mathbf{B}]$  – the matrix being defined according to formulas (9.55) or (9.56) depending on the frame (*Cartesian* or *cylindrical* frame). The *vector of stresses* according to the Hooke law is defined by the following relation:

$$\{\boldsymbol{\sigma}\} = [\mathbf{D}] \{\boldsymbol{\varepsilon}\} \quad (10.20)$$

where  $[\mathbf{D}]$  – matrix being defined according to formulas (9.49) or (9.52) depending on the frame. By substituting expression (10.18) by (10.17), we obtain equations of motion of a finite element, which are defined as follows:

$$[\mathbf{m}^{(e)}] \{\ddot{\mathbf{u}}^{(e)}\} + [\mathbf{c}^{(e)}] \{\dot{\mathbf{u}}^{(e)}\} + [\mathbf{k}^{(e)}] \{\mathbf{u}^{(e)}\} = \{\mathbf{f}^{(e)}\}, \quad (10.21)$$

$$\text{where } [\mathbf{m}^{(e)}] = \int_{V^{(e)}} \rho [\mathbf{N}]^T [\mathbf{N}] dV; \quad [\mathbf{c}^{(e)}] = \int_{V^{(e)}} \xi [\mathbf{N}]^T [\mathbf{N}] dV; \quad [\mathbf{k}^{(e)}] = \int_{V^{(e)}} [\mathbf{B}]^T [\mathbf{D}] [\mathbf{B}] dV;$$

$$\{\mathbf{f}^{(e)}\} = \int_{V^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_v\} dV + \int_{S^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_s\} dS + \sum_i \{\mathbf{f}_k\}_i. \quad (10.22)$$

If *damping coefficient*  $\xi$  is unknown, *damping matrix* is defined as follows:

$$[\mathbf{c}^{(e)}] = \alpha_1 [\mathbf{k}^{(e)}] + \alpha_2 [\mathbf{m}^{(e)}] \quad (10.23)$$

where:  $\alpha_1$ ,  $\alpha_2$  – coefficients being defined over the formulas (10.5). The *common system of dynamic balance* of an elastic body is formed from the equations system of finite elements (10.21) and accepts the following form:

$$[M]\{\ddot{u}\} + [C]\{\dot{u}\} + [K]\{u\} = \{F(t)\} \quad (10.24)$$

To solve the motion equations system (10.24), it is necessary to define *initial conditions* as follows:

$$\{u(x, t = 0)\} = \{u_0(x)\}; \quad \{\dot{u}(x, t = 0)\} = \{\dot{u}_0(x)\} \quad (10.25)$$

and *boundary conditions* as follows:

$$\{u(t, x = x_0)\} = \{u_0(t)\} \quad (10.26)$$

The equations system (10.24) are integrated by the method of *trapezoids*. The vectors of *displacements* and *velocities* at the moment  $t + \tau$  are determined according to the *trapezoid rule* as follows:

$$\{u_{t+\tau}\} = \{u_t\} + \frac{\tau}{2} (\{\dot{u}_t\} + \{\dot{u}_{t+\tau}\}); \quad (10.27)$$

$$\{\dot{u}_{t+\tau}\} = \{\dot{u}_t\} + \frac{\tau}{2} (\{\ddot{u}_t\} + \{\ddot{u}_{t+\tau}\}) \quad (10.28)$$

where:  $\tau$  - an integration step. Following these expressions, it is possible to define a *vector of accelerations*  $\{\ddot{u}_{t+\tau}\}$  via a *vector of displacements* at the moment  $t + \tau$ , namely:

$$\{\ddot{u}_{t+\tau}\} = \frac{4}{\tau^2} (\{u_{t+\tau}\} - \{u_t\}) - \frac{4}{\tau} \{\dot{u}_t\} - \{\ddot{u}_t\} \quad (10.29)$$

By substituting a *vector of accelerations* into the expression (10.28), it is possible to determine a *vector of velocities* at the moment  $t + \tau$ , namely:

$$\{\dot{u}_{t+\tau}\} = \frac{2}{\tau} (\{u_{t+\tau}\} - \{u_t\}) - \{\dot{u}_t\} \quad (10.30)$$

By substituting expressions (10.29) and (10.30) into the motion equations system it is possible to determine and a *displacements vector* at the moment  $t + \tau$ , namely:

$$\left( \frac{4}{\tau^2} [M] + \frac{2}{\tau} [C] + [K] \right) \{u_{t+\tau}\} = \{F_{t+\tau}\} + [M] \left( \frac{4}{\tau^2} \{u_t\} + \frac{4}{\tau} \{\dot{u}_t\} + \{\ddot{u}_t\} \right) + [C] \left( \frac{2}{\tau} \{u_t\} + \{\dot{u}_t\} \right) \quad (10.31)$$

The *vectors of accelerations* and *velocities* at the moment  $t + \tau$  are determined from expressions (10.29) and (10.30). While *natural frequencies* and *forms of free oscillations* are determined from the solution of the *eigenvalues problem*, namely:

$$([K] + i\omega[C] - \omega^2[M])\{U\} = 0 \quad (10.32)$$

where  $\omega$  - *natural frequency*. Minimum natural frequencies and forms without taking of a damping matrix into consideration are being determined by the method of iterations in a subspace (see section 10.1).

### 10.2.2. Input data for the solution of the problem

To solve the *dynamic problem* of the theory of elasticity at *dynamic loadings*, the *Dynamic\_plane* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable F. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelelem*), the number of nodes (*npoint*), the number of groups of finite elements (*ngr*) and the number of the known boundary conditions (*nbond*) are coded. Those elements, which have identical values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$  of a *damping*, which define a *damping matrix* are included into the corresponding group of finite elements.

In the *second* line, the number of nodes, in which additional coefficients of stiffness and dampings (*nst*) are given; the number of nodes, to which the external forces (*nforc*) are applied; the number of

finite elements, onto which the distributed loadings ( $nq$ ) are acted, and a code of the plane tense state ( $kstr$ ) are recorded;  $kstr=0$  for the *plain stress state* and  $kstr=1$  for the *plane strain*.

In the *third* line, a print code of intermediate results ( $kprint$ ), a problem type ( $ngama$ ), a code of the solution of system of the algebraic equations ( $ksolve$ ), the number of iterations ( $niter$ ), which is necessary for the solution of an equations system, and a precision of solution ( $toler$ ). If  $kprint=0$ , then the intermediate results are not printed; otherwise, they are printed out. If the problem is solved in the *cartesian* frame, then  $ngama=0$ , if  $ngama=1$  - in the *cylindrical* frame. If parameter  $ksolve=0$ , then to solve, the `'solve'` function of the *Maple*-language is used; otherwise, the equations system is solved by the method of *conjugate gradients*.

In the *fourth* line, the number of integration steps ( $ntime$ ), and an integration step ( $dtime$ ) are coded. In the *fifth* line, the number of eigenvalues ( $neigen$ ); the number of iterations ( $nitero$ ); precision of definition of eigenvalues and vectors ( $tolero$ ), are coded. If definitions of eigenvalues and vectors are not required, then relation  $neigen=0$  is supposed. In the *sixth* line, the number of degrees of freedom, for which into the file the values of results are recorded ( $nout$ ); and the number indicating through how much of integration steps the results of evaluations are recorded ( $nstep$ ) are coded.

In each subsequent  $nelem$  lines, the number of a finite element and the numbers of nodes of this element {array  $Mtop(nelem, nnode)$ , where  $nnode$  - number of nodes of the finite element, i.e.  $nnode=4$ } are coded, and also the number of the group, to which this finite element {array  $Mgr(nelem)$ } belongs, and the number of a body, to which the given finite element {array  $Mkel(nelem)$ } belongs. After arrays  $Mtop$  and  $Mgr$ , the elements of  $Lbond(nbond)$  array are coded line by line. Into each line of the file the line number and element of  $Lbond$  array are recorded. Into each line of  $Lbond$  array the numbers of nodes, where boundary conditions are known, are coded.

After array  $Lbond$ , the elements of  $Lst(nst)$  array are coded line by line. Into each line of the file the line number and element of  $Lst$  array are recorded. Into each line of  $Lst$  array the number of a node, where additional coefficients of stiffness and dampings are given, are recorded. After array  $Lst$ , the elements of  $Lforc(nforc)$  array are coded line by line. Into each line of the file a line number and the element of  $Lforc$  array are recorded. Into each line of  $Lforc$  array the numbers of nodes of the finite elements, where external forces affect, are recorded.

After array  $Lforc$ , the elements of  $Lq(nq, 3)$  array are coded line by line. Into each line of the file, a line number and the element of  $Lq$  array are recorded. Into each line of  $Lq$  array, the number of a finite element and also two nodes of its side, onto which the distributed loading affects, are recorded. After array  $Lq$ , the elements of  $Lout(nout)$  array are coded line by line. Into each line of the file a line number and the elements of  $Lout$  array are recorded. Into the  $Lout$  array, the numbers of degrees of freedom, for which the values of results are recorded into the file, are coded.

Behind array  $Lout$ , the elements of  $Coord(npoin, ndime)$  array, where  $ndime$  - dimensionality of the problem ( $ndime=2$ ), are recorded line by line into the data file. Into the *first* column of the  $Coord$  array the  $x$ -coordinates are recorded; while  $y$ -coordinates of nodes of finite elements are recorded into the *second*. Into each line of the file the number of a node and its  $(x, y)$ -coordinates are recorded. Behind array  $Coord$ , the elements of  $Bond(nbond)$  array are recorded line by line. Into each line of the datafile, a line number of  $Bond$  array and a value of boundary condition are recorded. The lines of  $Bond$  array should correspond to the lines of  $Lbond$  array.

Behind  $Bond$  array, the elements of  $Pgr(ngr, 5)$  array are coded line by line. Into each line of the datafile a line number of  $Pgr$  array, values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$  of *damping*, which define a *damping matrix*, are recorded. The line number of  $Pgr$  array should strictly correspond to the number of the group of finite elements. After array  $Pgr$ , the elements of  $Ast(nst, 4)$  array are recorded line by line. Into each line of the datafile, a line number of  $Ast$  array, two values of additional coefficients of stiffness, and two

values of additional coefficients of damping, which correspond to each degree of freedom in a node, are recorded. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After array **Ast**, the elements of **Forc(nforc, 8)** array are recorded line by line. Into each line, a line number of **Forc** array and two groups of coefficients  $\mathbf{a}_i$  ( $i=0 \dots 3$ ), which enter a definition of concentrated forces  $\mathbf{f}_k(\mathbf{t}) = [\mathbf{a}_0 + \mathbf{a}_1 \sin(\mathbf{a}_2 \mathbf{t})] \mathbf{e}^{\mathbf{a}_3 \mathbf{t}}$  ( $k=1 \dots 3$ ), acting in the direction of each degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array.

After array **Forc**, the elements of **Fq(nq, 6)** arrays are recorded line by line. Into each line, a line number of array **Fq** and the values of coefficients  $\mathbf{q}_1, \mathbf{q}_2, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$ , by which surface forces are computed as follows:  $\mathbf{f}_s(\mathbf{r}, \mathbf{s}, \mathbf{t}) = [\mathbf{q}_i \mathbf{N}_i(\mathbf{r}, \mathbf{s}) + \mathbf{q}_j \mathbf{N}_j(\mathbf{r}, \mathbf{s})] [\mathbf{b}_0 + \mathbf{b}_1 \sin(\mathbf{b}_2 \mathbf{t})] \mathbf{e}^{\mathbf{b}_3 \mathbf{t}}$  ( $\mathbf{q}_i, \mathbf{q}_j$  - values of surface forces in nodes  $\mathbf{i}$  and  $\mathbf{j}$ ), are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. When reading information from the datafile, these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nst, nforc, nq, kstr, kprint, ngama, ksolve, niter, toler, ntime, dtime, neigen, nitero, tolero, nout, nstep*  
 Text line \*  
 Arrays *Mtop(nelem, nmode), Mgr(nelem)*  
 Text line \*  
 Array *Lbond(nbond)*  
 Text line \*  
 Array *Lst(nst)*  
 Text line \*  
 Array *Lforc(nforc)*  
 Text line \*  
 Array *Lq(nq, 3)*  
 Text line \*  
 Array *Coord(npoin, ndime)*  
 Text line \*  
 Array *Bond(nbond)*  
 Text line \*  
 Array *Pgr(ngr, 5)*  
 Text line \*  
 Array *Ast(nst, 4)*  
 Text line \*  
 Array *Forc(nforc, 8)*  
 Text line \*  
 Array *Fq(nq, 6)*

### 10.2.3. Brief description of the *Dynamic\_plane* program solving the problem

The *Dynamic\_plane* program was programmed in the *Maple*-language; it consists of the basic program and 62 procedures. All procedures can be divided into *three* groups: procedures for data entry, calculation and output of results. Memory size necessary for the solution of a concrete problem and time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of *displacements, velocities* and *accelerations* of nodes of finite

elements in time dependence and *natural frequencies* and *forms*. Calculation results are output on the monitor and are recorded into a file; therefore, a *qualifier* of a target file must be ascribed to variable *file\_rez1* of the program. Into the file *file\_rez1* values of *displacements*, *velocities* and *accelerations* of nodes of finite elements (*in a time dependence*) in degrees of freedom indicated in array **Lout(nout)** and *natural frequencies* and *forms* are recorded.

### 10.2.4. An example of the use of the Maple-program Dynamic\_plane

As an example of application of the program, the motion of a box-like construction made from *aluminium* is considered; thickness of the construction is 0.05 m. (fig. 10.3). The short-term surface force  $f_s(r, s, t)$  of the following view affects onto construction  $t$ :  $f_s(r, s, t) = [q_i N_i(r, s) + q_j N_j(r, s)]b_0 + b_1 \sin(b_2 t)]e^{b_3 t}$ , where:  $b_0 = 1 \frac{N}{m}$ ,  $b_1 = b_2 = 0$ ,  $b_3 = -7000 \text{ s}^{-1}$  and concentrated force  $f_k = -10^2 \text{ N}$ .

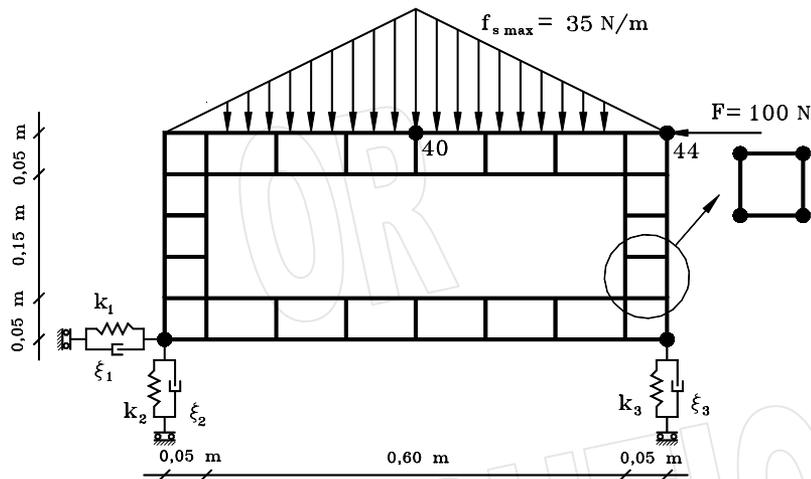


Fig. 10.3. The calculated scheme of a box-like construction

Input data for the test example: number of finite elements  $n_{elem}=22$ , the number of nodes  $n_{poin}=44$ , the number of groups of finite elements  $n_{gr}=1$ , the number of boundary conditions  $n_{bond}=1$ ,  $k_{print}=0$ ,  $n_q=8$ ,  $n_{st}=2$ ,  $n_{forc}=1$ ,  $k_{solve}=0$ ,  $n_{iter}=300$ ,  $t_{oler}=10^{-9}$ ,  $n_{gamma}=0$ ,  $n_{kd}=1$ ,  $n_{time}=100$ ,  $d_{time}=10^{-5} \text{ s}$ ,  $n_{eigen}=5$ ,  $n_{iterv}=100$ ,  $n_{out}=2$ ,  $n_{step}=1$ ,  $k_{str}=0$ .

Parameters of the first group:  $\rho = 2700 \frac{\text{kg}}{\text{m}^3}$ ;  $E = 70 \text{ GPa}$ ;  $\mu = 0,25$ ;  $\alpha_1 = \alpha_2 = 0$

Coefficients of additional stiffness and damping:  $k_1 = k_2 = k_3 = 10^6 \frac{\text{N}}{\text{m}}$ ;  $\xi_1 = \xi_2 = \xi_3 = 10^{-3} \frac{\text{N}\cdot\text{s}}{\text{m}}$

#### Results of calculation over the test example:

##### Natural frequencies (Hz):

- 1  $w=10.082576e+01$
- 2  $w=1.042424e+02$
- 3  $w=1.544430e+02$
- 4  $w=6.629204e+02$
- 5  $w=7.855101e+02$

##### Natural forms:

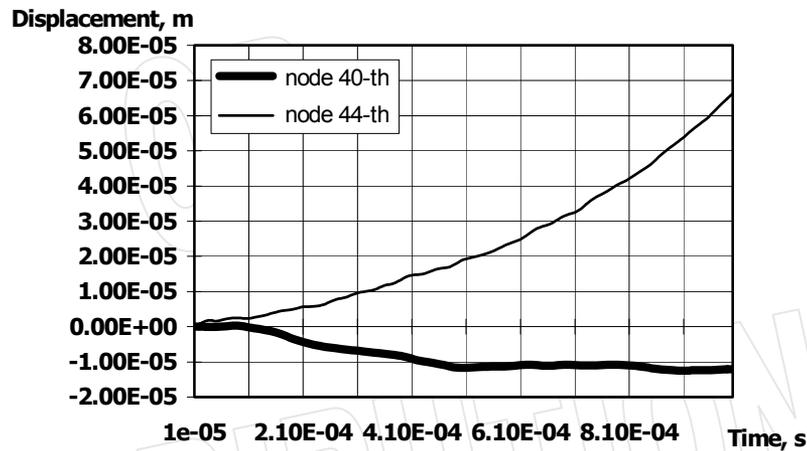
1	-6.700339e-06	-2.540384e-08	1.096942e-06	1.029982e-08	6.845697e-08
2	-1.535971e-06	-3.398411e-06	-2.394439e-06	-1.082814e-08	5.973090e-09

3	-6.714254e-06	-2.558167e-08	1.100027e-06	1.211132e-08	6.596320e-08
4	-1.275486e-06	-3.419510e-06	-2.064406e-06	1.225458e-08	2.668570e-08
5	-6.971102e-06	-8.745572e-09	7.653646e-07	-1.267753e-08	4.750100e-08
6	-1.538027e-06	-3.405883e-06	-2.398989e-06	-8.982261e-09	3.727903e-09
7	-6.967229e-06	-6.984760e-09	7.652176e-07	-1.371339e-08	4.972239e-08
8	-1.273646e-06	-3.418057e-06	-2.064027e-06	1.091343e-08	2.767103e-08
9	-6.733334e-06	-2.479540e-08	1.104666e-06	1.382729e-08	5.914546e-08
10	-7.961512e-07	-3.461336e-06	-1.388975e-06	6.994296e-08	4.296853e-08
11	-6.747270e-06	-1.839203e-08	1.109138e-06	8.692097e-09	5.431788e-08
12	-3.581312e-07	-3.496796e-06	-7.004674e-07	1.197394e-07	2.977861e-08
13	-6.756558e-06	-9.456602e-09	1.111558e-06	-1.427796e-10	5.251140e-08
14	5.653418e-08	-3.508589e-06	-2.238227e-09	1.389952e-07	3.096743e-10
15	-6.761151e-06	-5.052633e-10	1.111402e-06	-9.002925e-09	5.440748e-08
16	4.642875e-07	-3.491903e-06	6.971144e-07	1.201680e-07	-2.922647e-08
17	-6.761111e-06	5.946503e-09	1.109196e-06	-1.420685e-08	5.934730e-08
18	8.813718e-07	-3.451439e-06	1.389024e-06	7.059637e-08	-4.256901e-08
19	-6.755612e-06	6.823088e-09	1.106774e-06	-1.259208e-08	6.631824e-08
20	1.326941e-06	-3.404356e-06	2.069946e-06	1.275554e-08	-2.638755e-08
21	-6.961565e-06	-4.113225e-09	7.637432e-07	-1.466300e-08	5.899921e-08
22	-7.955981e-07	-3.462007e-06	-1.389308e-06	7.023469e-08	4.254248e-08
23	-6.959526e-06	-6.041435e-09	7.620468e-07	-9.380303e-09	6.574720e-08
24	-3.573671e-07	-3.496968e-06	-7.005846e-07	1.196478e-07	2.976156e-08
25	-6.961210e-06	-1.067649e-08	7.616529e-07	-2.558820e-10	6.820910e-08
26	5.719811e-08	-3.508902e-06	-2.346610e-09	1.390112e-07	3.021660e-10
27	-6.966463e-06	-1.532380e-08	7.631824e-07	8.896117e-09	6.589704e-08
28	4.647671e-07	-3.492077e-06	6.970284e-07	1.200767e-07	-2.922429e-08
29	-6.975529e-06	-1.729161e-08	7.660284e-07	1.425673e-08	5.927404e-08
30	8.823605e-07	-3.452108e-06	1.389106e-06	7.088431e-08	-4.215787e-08
31	-6.988045e-06	-1.448861e-08	7.686204e-07	1.341633e-08	5.007630e-08
32	1.325371e-06	-3.402917e-06	2.069520e-06	1.142693e-08	-2.738630e-08
33	-6.753042e-06	6.706132e-09	1.105542e-06	-1.083014e-08	6.894016e-08
34	1.562834e-06	-3.380508e-06	2.403968e-06	-1.057368e-08	-5.615861e-09
35	-6.991317e-06	-1.275623e-08	7.686654e-07	1.240993e-08	4.784408e-08
36	1.568537e-06	-3.387939e-06	2.407947e-06	-8.749744e-09	-3.394167e-09
37	-7.238022e-06	-4.551698e-09	4.270935e-07	-2.672815e-08	1.820687e-08
38	-1.540718e-06	-3.413483e-06	-2.402460e-06	-3.080609e-09	-6.562833e-10
39	-7.238634e-06	-4.533499e-09	4.274176e-07	-2.631801e-08	1.773918e-08
40	-1.272769e-06	-3.413436e-06	-2.062678e-06	3.481267e-09	3.168098e-08

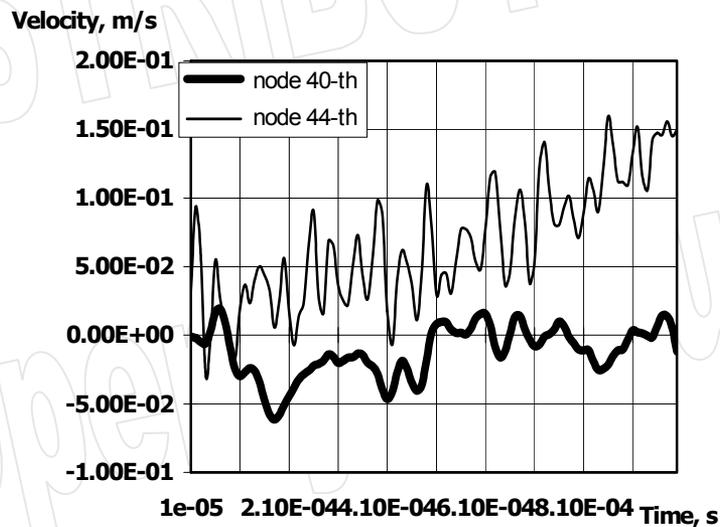
41	-7.506443e-06	-9.276544e-09	8.536292e-08	-2.653654e-08	-1.720118e-08
42	-1.538365e-06	-3.416327e-06	-2.404614e-06	2.722807e-09	-6.599699e-10
43	-7.505735e-06	-8.636375e-09	8.559097e-08	-2.612940e-08	-1.672883e-08
44	-1.276094e-06	-3.413027e-06	-2.062360e-06	-4.213948e-09	3.167922e-08
45	-7.762490e-06	-1.137951e-08	-2.574818e-07	-1.212308e-08	-4.648646e-08
46	-1.531697e-06	-3.414363e-06	-2.405256e-06	8.615673e-09	3.725138e-09
47	-7.765041e-06	-1.289670e-08	-2.575620e-07	-1.314537e-08	-4.873077e-08
48	-1.283199e-06	-3.416778e-06	-2.063071e-06	-1.163826e-08	2.766438e-08
49	-7.244042e-06	-1.982935e-08	4.282874e-07	2.639431e-08	1.805347e-08
50	1.319328e-06	-3.398326e-06	2.069015e-06	4.036930e-09	-3.140374e-08
51	-7.243536e-06	-1.980739e-08	4.279793e-07	2.679824e-08	1.852480e-08
52	1.576284e-06	-3.395490e-06	2.410607e-06	-2.894442e-09	9.854825e-10
53	-7.504646e-06	-1.862504e-08	8.538577e-08	2.659400e-08	-1.648035e-08
54	1.319280e-06	-3.397905e-06	2.069263e-06	-3.663206e-09	-3.144018e-08
55	-7.505355e-06	-1.799109e-08	8.515467e-08	2.700676e-08	-1.694794e-08
56	1.577344e-06	-3.398335e-06	2.412202e-06	2.914503e-09	1.010091e-09
57	-7.762520e-06	-1.722323e-08	-2.579907e-07	1.396021e-08	-4.862363e-08
58	1.324703e-06	-3.401603e-06	2.070277e-06	-1.113814e-08	-2.748725e-08
59	-7.759873e-06	-1.871933e-08	-2.579192e-07	1.291150e-08	-4.639418e-08
60	1.572345e-06	-3.396416e-06	2.412554e-06	8.862491e-09	-3.327032e-09
61	-8.006875e-06	-7.371606e-09	-5.997812e-07	1.121849e-08	-6.752381e-08
62	-1.528794e-06	-3.413081e-06	-2.405361e-06	1.047179e-08	5.962196e-09
63	-8.003590e-06	-5.825801e-09	-5.997142e-07	1.299050e-08	-6.489938e-08
64	-1.284742e-06	-3.417656e-06	-2.063248e-06	-1.297615e-08	2.666468e-08
65	-7.774478e-06	-1.778755e-08	-2.575008e-07	-1.403049e-08	-5.798455e-08
66	-8.265680e-07	-3.441054e-06	-1.379125e-06	-7.143831e-08	4.222865e-08
67	-7.780113e-06	-1.778592e-08	-2.564089e-07	-8.668824e-09	-6.465082e-08
68	-3.959880e-07	-3.464395e-06	-6.900017e-07	-1.209487e-07	2.905623e-08
69	-7.781650e-06	-1.512979e-08	-2.559041e-07	5.127084e-10	-6.699345e-08
70	2.087766e-08	-3.472373e-06	3.801057e-09	-1.400221e-07	-6.374537e-10
71	-7.779296e-06	-1.245930e-08	-2.565671e-07	9.667340e-09	-6.455878e-08
72	4.376943e-07	-3.459469e-06	6.975438e-07	-1.205487e-07	-3.010827e-08
73	-7.772838e-06	-1.241162e-08	-2.578035e-07	1.495266e-08	-5.784781e-08
74	8.681242e-07	-3.431103e-06	1.386519e-06	-7.082769e-08	-4.271390e-08
75	-7.995665e-06	-4.871793e-09	-6.005915e-07	1.460811e-08	-5.794126e-08
76	-8.260731e-07	-3.440512e-06	-1.379022e-06	-7.114453e-08	4.264606e-08
77	-7.991152e-06	-9.350042e-09	-6.023034e-07	9.351362e-09	-5.305450e-08
78	-3.960528e-07	-3.464187e-06	-6.899864e-07	-1.210374e-07	2.906492e-08

79	-7.989334e-06	-1.635866e-08	-6.030882e-07	4.073172e-10	-5.124808e-08
80	2.082662e-08	-3.472071e-06	3.808873e-09	-1.400023e-07	-6.365179e-10
81	-7.990286e-06	-2.337677e-08	-6.024061e-07	-8.515605e-09	-5.317068e-08
82	4.376474e-07	-3.459261e-06	6.975462e-07	-1.206373e-07	-3.011621e-08
83	-7.993954e-06	-2.788419e-08	-6.008077e-07	-1.371572e-08	-5.812759e-08
84	8.675520e-07	-3.430566e-06	1.386426e-06	-7.052901e-08	-4.312658e-08
85	-8.001034e-06	-2.698872e-08	-6.000745e-07	-1.201574e-08	-6.507451e-08
86	1.326022e-06	-3.402469e-06	2.070496e-06	-1.248829e-08	-2.649997e-08
87	-8.004106e-06	-2.546869e-08	-6.001848e-07	-1.021241e-08	-6.767996e-08
88	1.569733e-06	-3.395156e-06	2.412604e-06	1.074574e-08	-5.544962e-09

On *fig. 10.4* the dependences of vertical displacements (a) and velocities (b) of nodes with numbers 40 and 44 of the finite element in time are represented.



(a)



(b)

Fig. 10.4. Dependences of vertical displacements (a) and velocities (b) of nodes with numbers 40 and 44 of the finite element in time

The *Dynamic\_plane* program is intended for the solution of the *plane problem* of the theory of elasticity at *dynamic loadings*. Natural frequencies and forms are determined as well as the system of dynamic balance is integrated in time. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in PROBLEMS directory of archive attached to the book in files *Mb7\_2\_new.mws*, *Mb7\_2.dat* and *Mb7\_2\_1.rez* accordingly.

### 10.3. Dynamic problem of shells of an arbitrary contour

The major number of objects of mechanical engineering constitute *shells* of an arbitrary contour. To calculate *thin shells* of an arbitrary contour, the FEM is used effectively. A *shell* can be idealized by *plurality of finite elements* of diversified forms at various guesses in the attitude of a tense state. Now *isoparametric finite elements* are used rather widely. *Shell-similar* objects are being exposed to loadings of various kinds; however distributed loading can be accepted on shell surface for basic loading, for example, internal pressure of some medium located in the shell. By taking into account the aforesaid, calculation of shells of an arbitrary contour has applied importance for the solution of manifold engineering problems. In the given section, the *dynamic problem of shells* is considered. The purpose of research is the evaluation of *natural frequency* and numerical integration of equations of motion of *shells of an arbitrary contour*.

#### 10.3.1. The calculated equations of the shells' problem

To calculate shells of an arbitrary contour, the following *two-dimensional quadratic nine-nodal isoparametric finite element* is used (fig. 10.5):

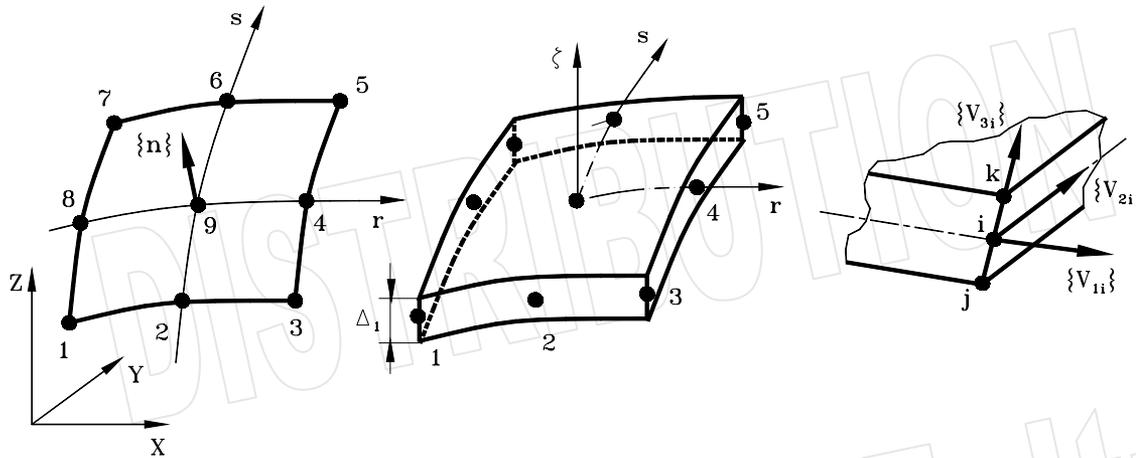


Fig. 10.5. Two-dimensional quadratic nine-nodal isoparametric finite element

The *functions of the form* for the given finite element are as follows:

$$\begin{aligned}
 N_1(r, s) &= \frac{1}{4}(1-r)(1-s) - \frac{1}{2}N_2(r, s) - \frac{1}{2}N_8(r, s) - \frac{1}{4}N_9(r, s); & N_2(r, s) &= \frac{1}{2}(1-r^2)(1-s) - \frac{1}{2}N_9(r, s); \\
 N_3(r, s) &= \frac{1}{4}(1+r)(1-s) - \frac{1}{2}N_2(r, s) - \frac{1}{2}N_4(r, s) - \frac{1}{4}N_9(r, s); & N_4(r, s) &= \frac{1}{2}(1-s^2)(1+r) - \frac{1}{2}N_9(r, s); \\
 N_5(r, s) &= \frac{1}{4}(1+r)(1+s) - \frac{1}{2}N_4(r, s) - \frac{1}{2}N_6(r, s) - \frac{1}{4}N_9(r, s); & N_6(r, s) &= \frac{1}{2}(1-r^2)(1+s) - \frac{1}{2}N_9(r, s); \\
 N_7(r, s) &= \frac{1}{2}(1-r^2)(1+s) - \frac{1}{2}N_9(r, s); & N_8(r, s) &= \frac{1}{2}(1-s^2)(1-r) - \frac{1}{2}N_9(r, s); & N_9(r, s) &= (1-r^2)(1-s^2).
 \end{aligned}
 \tag{10.33}$$

It is supposed that *tense state* in a medial plane of the element can be described by the *plane problem of the theory of elasticity*; while *tense state* arising at an arcuation of the element can be described by the *theory of arcuation of tough plates*.

As the basic *nodal unknown displacements* in each node of the finite element, three linear  $\mathbf{u}_i, \mathbf{v}_i, \mathbf{w}_i$  and two angular  $\alpha_i, \beta_i$  displacements are introduced. Thus, the two-dimensional isoparametric finite element has 45 degrees of freedom. The *thickness* of a finite element in each node is designated as  $\Delta_i$  ( $i=1..8$ ). In nodes which lie on a medial plane of a shell the *vector*  $\{\mathbf{V}_{3i}\}$ , perpendicular to this plane, is introduced, namely:

$$\{\mathbf{V}_{3i}\}^T = \Delta_i [l_{3i} \ m_{3i} \ n_{3i}] \quad (10.34)$$

where  $l_{3i}, m_{3i}, n_{3i}$  - *direction cosines* of this vector (*straight line i-j-k on the fig. 10.5*), and:

$$l_{3i} = \frac{x_k - x_i}{\Delta_i}; \ l_{3i} = \frac{x_k - x_i}{\Delta_i}; \ m_{3i} = \frac{y_k - y_i}{\Delta_i}; \ n_{3i} = \frac{z_k - z_i}{\Delta_i}; \ \text{where: } \mathbf{x}_i, \mathbf{y}_i, \quad (10.35)$$

where  $\mathbf{z}_i$  - *global coordinates* of the corresponding point. *Global coordinates* of some point of the finite element are defined from the following relations:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \sum_{i=1}^9 N_i(r,s) \begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} + \sum_{i=1}^9 N_i(r,s) \zeta \frac{\Delta_i}{2} \begin{Bmatrix} l_{3i} \\ m_{3i} \\ n_{3i} \end{Bmatrix}, \quad (10.36)$$

where  $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$  - *global coordinates* of a shell midpoint and  $x_i=x_k-x_j; y_i=y_k-y_j; z_i=z_k-z_j$ .

In each node of the finite element three mutually perpendicular vectors  $\mathbf{V}_{1i}, \mathbf{V}_{2i}$  and  $\mathbf{V}_{3i}$  are additionally introduced, which are *tangential* to the medial plane. The components of the vector  $\mathbf{V}_{1i}$  can be defined as follows:

$$\{\mathbf{V}_{1i}\} = \Delta r \sum_{i=1}^9 \begin{Bmatrix} \frac{\partial N_i}{\partial r} x_i \\ \frac{\partial N_i}{\partial r} y_i \\ \frac{\partial N_i}{\partial r} z_i \end{Bmatrix}, \quad (10.37)$$

where  $\Delta r$  - *small increment* of *r-coordinate*. In addition, the *vector*  $\mathbf{V}_{2i}$  is determined from the following condition:

$$\{\mathbf{V}_{2i}\} = \{\mathbf{V}_{3i}\} \times \{\mathbf{V}_{1i}\} = \begin{Bmatrix} m_{3i}n_{1i} - n_{3i}m_{1i} \\ n_{3i}l_{1i} - l_{3i}n_{1i} \\ l_{3i}m_{1i} - m_{3i}l_{1i} \end{Bmatrix} \quad (10.38)$$

The *vector of displacements* in any point of the finite element is as follows:

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix} = \sum_{i=1}^9 N_i(r,s) \begin{Bmatrix} u_i \\ v_i \\ w_i \end{Bmatrix} + \sum_{i=1}^9 N_i(r,s) \zeta \frac{\Delta_i}{2} [\mathbf{a}_i] \begin{Bmatrix} \alpha_i \\ \beta_i \end{Bmatrix}; \quad (10.39)$$

$$\text{where } [\mathbf{a}_i] - \text{matrix of direction cosines, and } [\mathbf{a}_i] = [-\{\mathbf{V}_{2i}\} \ \{\mathbf{V}_{1i}\}]. \quad (10.40)$$

Then the *vector of strain* is defined by the following relations:

$$\{\varepsilon\}^T = [\varepsilon_{xx} \quad \varepsilon_{yy} \quad \varepsilon_{zz} \quad \varepsilon_{xy} \quad \varepsilon_{yz} \quad \varepsilon_{zx}] = [A]\{e_{xyz}\},$$

where  $\{e_{xyz}\}^T = \left[ \frac{\partial u}{\partial x} \quad \frac{\partial u}{\partial y} \quad \frac{\partial u}{\partial z} \quad \frac{\partial v}{\partial x} \quad \frac{\partial v}{\partial y} \quad \frac{\partial v}{\partial z} \quad \frac{\partial w}{\partial x} \quad \frac{\partial w}{\partial y} \quad \frac{\partial w}{\partial z} \right];$  (10.41)

$$[A] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Vector  $\{e_{xyz}\}$  can be expressed by the following relation:

$$\{e_{xyz}\} = \begin{bmatrix} [J]^{-1} & 0 & 0 \\ 0 & [J]^{-1} & 0 \\ 0 & 0 & [J]^{-1} \end{bmatrix} \{e_{rs\zeta}\},$$

where  $\{e_{rs\zeta}\}^T = \left[ \frac{\partial u}{\partial r} \quad \frac{\partial u}{\partial s} \quad \frac{\partial u}{\partial \zeta} \quad \frac{\partial v}{\partial r} \quad \frac{\partial v}{\partial s} \quad \frac{\partial v}{\partial \zeta} \quad \frac{\partial w}{\partial r} \quad \frac{\partial w}{\partial s} \quad \frac{\partial w}{\partial \zeta} \right];$  (10.42)

$[J]$  - Jacobi matrix and  $[J] = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{bmatrix};$  (10.43)

$$\frac{\partial x}{\partial r} = \sum_{i=1}^9 \frac{\partial N_i}{\partial r} \left( x_i + \frac{\zeta \Delta_i l_{3i}}{2} \right); \quad \frac{\partial x}{\partial s} = \sum_{i=1}^9 \frac{\partial N_i}{\partial s} \left( x_i + \frac{\zeta \Delta_i l_{3i}}{2} \right); \quad \frac{\partial x}{\partial \zeta} = \sum_{i=1}^9 N_i \frac{\Delta_i l_{3i}}{2}$$
 (10.44)

Then, vector  $\{e_{rs\zeta}\}$  can be expressed as follows:

$$\{e_{rs\zeta}\} = \sum_{i=1}^9 [H]\{\Phi_i\},$$
 (10.45)

where:  $\{\Phi_i\}^T = [u_i \quad v_i \quad w_i \quad \alpha_i \quad \beta_i];$

$$[H] = \begin{bmatrix} N_{i,r} & 0 & 0 & -\frac{\zeta \Delta_i N_{i,r} l_{2i}}{2} & \frac{\zeta \Delta_i N_{i,r} l_{1i}}{2} \\ N_{i,s} & 0 & 0 & -\frac{\zeta \Delta_i N_{i,r} l_{2i}}{2} & \frac{\zeta \Delta_i N_{i,s} l_{1i}}{2} \\ 0 & 0 & 0 & -\frac{\Delta_i N_i l_{2i}}{2} & \frac{\Delta_i N_i l_{1i}}{2} \\ 0 & N_{i,r} & 0 & -\frac{\zeta \Delta_i N_{i,r} m_{2i}}{2} & \frac{\zeta \Delta_i N_{i,r} m_{1i}}{2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}; \quad N_{i,r} = \frac{\partial N_i}{\partial r}, \text{ etc.}$$
 (10.46)

Finally, the *vector of strains* is as follows:

$$\{\boldsymbol{\varepsilon}\} = \sum_{i=1}^9 [\mathbf{B}_i] \{\boldsymbol{\Phi}_i\} = [\mathbf{B}] \{\mathbf{u}^{(e)}\}, \quad (10.47)$$

where  $(6 \times 5)$  - dimensionality of the matrix  $[\mathbf{B}_i]$ ;  $\{\mathbf{u}^{(e)}\}$  - vector of nodal displacements and  $\{\mathbf{u}^{(e)}\}^T = [\{\boldsymbol{\Phi}_1\}^T \ \{\boldsymbol{\Phi}_2\}^T \ \dots \ \{\boldsymbol{\Phi}_9\}^T]$ . The dependence of the *vector of stresses*  $\{\boldsymbol{\sigma}\}$  from the *vector of strains* is defined according to the generalized Hooke law, namely:

$$\{\boldsymbol{\sigma}\} = [\mathbf{D}] \{\boldsymbol{\varepsilon}\} \quad (10.48)$$

The matrix  $[\mathbf{D}]$  for an *isotropic material* under condition  $\sigma_{zz} = 0$  is as follows:

$$[\mathbf{D}] = \begin{bmatrix} \underline{D}_{11} & \underline{D}_{12} & 0 & 0 & 0 & 0 \\ \underline{D}_{21} & \underline{D}_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \underline{D}_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \underline{D}_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \underline{D}_{66} \end{bmatrix}, \quad (10.49)$$

$$\text{where } \underline{D}_{11} = \underline{D}_{22} = \frac{\underline{D}_{12}}{\mu} = \frac{E}{1-\mu^2}; \quad \underline{D}_{44} = \underline{D}_{55} = \underline{D}_{66} = \frac{E}{2(1+\mu)}.$$

Following the aforesaid, the *equations system* of the dynamic balance for a shell finite element is:

$$[\mathbf{m}^{(e)}] \{\ddot{\mathbf{u}}^{(e)}\} + [\mathbf{c}^{(e)}] \{\dot{\mathbf{u}}^{(e)}\} + [\mathbf{k}^{(e)}] \{\mathbf{u}^{(e)}\} = \{\mathbf{f}^{(e)}\}, \quad (10.50)$$

where  $[\mathbf{m}^{(e)}]$ ,  $[\mathbf{c}^{(e)}]$ ,  $[\mathbf{k}^{(e)}]$  - matrixes of masses, a damping and stiffness;  $\{\mathbf{f}^{(e)}\}$  - vector of external forces. Then, the *stiffness matrix* of a finite element is defined as follows:

$$[\mathbf{k}^{(e)}] = \int_{V^{(e)}} [\mathbf{B}]^T [\mathbf{D}] [\mathbf{B}] \, dV, \quad (10.51)$$

where  $[\mathbf{D}] = [\mathbf{T}]^T [\mathbf{D}] [\mathbf{T}]$ ;  $[\mathbf{T}]$  - matrix of transformation of coordinates and

$$[\mathbf{T}] = \begin{bmatrix} l_1^2 & m_1^2 & n_1^2 & l_1 m_1 & m_1 n_1 & n_1 l_1 \\ l_2^2 & m_2^2 & n_2^2 & l_2 m_2 & m_2 n_2 & n_2 l_2 \\ l_3^2 & m_3^2 & n_3^2 & l_3 m_3 & m_3 n_3 & n_3 l_3 \\ 2l_1 l_2 & 2m_1 m_2 & 2n_1 n_2 & l_1 m_2 + l_2 m_1 & m_1 n_2 + m_2 n_1 & n_1 l_2 + n_2 l_1 \\ 2l_2 l_3 & 2m_2 m_3 & 2n_2 n_3 & l_2 m_3 + l_3 m_2 & m_2 n_3 + m_3 n_2 & n_2 l_3 + n_3 l_2 \\ 2l_3 l_1 & 2m_3 m_1 & 2n_3 n_1 & l_3 m_1 + l_1 m_3 & m_3 n_1 + m_1 n_3 & n_3 l_1 + n_1 l_3 \end{bmatrix} \quad (10.52)$$

The *matrix of masses* of a finite element is defined by the following relation:

$$[\mathbf{m}^{(e)}] = \int_{V^{(e)}} \rho [\mathbf{N}]^T [\mathbf{N}] \, dV, \quad (10.53)$$

$$\text{where } [\mathbf{N}] = \begin{bmatrix} N_1 & 0 & 0 & a_1 & b_1 & \dots & N_9 & 0 & 0 & a_9 & b_9 \\ 0 & N_1 & 0 & c_1 & d_1 & \dots & 0 & N_9 & 0 & c_9 & d_9 \\ 0 & 0 & N_1 & g_1 & h_1 & \dots & 0 & 0 & N_9 & g_9 & h_9 \end{bmatrix}; \quad (10.54)$$

$$a_i = -N_i \zeta \frac{\Delta_i}{2} l_{2i}; \quad b_i = N_i \zeta \frac{\Delta_i}{2} l_{1i}; \quad c_i = -N_i \zeta \frac{\Delta_i}{2} m_{2i}; \quad d_i = N_i \zeta \frac{\Delta_i}{2} m_{1i}; \quad g_i = -N_i \zeta \frac{\Delta_i}{2} n_{2i}; \quad h_i = N_i \zeta \frac{\Delta_i}{2} n_{1i}$$

The matrix of a damping of a finite element is defined by the following relations:

$$[\mathbf{c}^{(e)}] = \alpha_1 [\mathbf{k}^{(e)}] + \alpha_2 [\mathbf{m}^{(e)}], \text{ where } \alpha_1, \alpha_2 - \text{known coefficients} \quad (10.55)$$

The vector of external forces of a finite element consists of vectors of volumetric, surface and concentrated forces, namely:

$$\{\mathbf{f}^{(e)}\} = \{\mathbf{f}_v^{(e)}\} + \{\mathbf{f}_s^{(e)}\} + \{\mathbf{f}_k^{(e)}\}, \quad (10.56)$$

$$\text{where } \{\mathbf{f}_v^{(e)}\} = \int_{V^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_v\} dV; \quad \{\mathbf{f}_s^{(e)}\} = \int_{S^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_s\} dS; \quad (10.57)$$

$$\{\mathbf{f}_v\}^T = [f_{vx} \quad f_{vy} \quad f_{vz}]; \quad \{\mathbf{f}_s\}^T = [f_{sx} \quad f_{sy} \quad f_{sz}]. \quad (10.58)$$

If the distributed loading  $\mathbf{p}(\mathbf{r}, \mathbf{s})$  acts perpendicularly to the surface of the finite element, the vector of surface forces is defined by following relation:

$$\{\mathbf{f}_s^{(e)}\} = \int_{S^{(e)}} [\mathbf{N}]^T \mathbf{p}(\mathbf{r}, \mathbf{s}) \{\mathbf{n}\} dS, \quad (10.59)$$

where  $\{\mathbf{n}\}^T = [n_x \quad n_y \quad n_z]$  - normal vector to a surface of a finite element and

$$n_x = \frac{\left(\frac{\partial y}{\partial r} \frac{\partial z}{\partial s} - \frac{\partial z}{\partial r} \frac{\partial y}{\partial s}\right)}{|\{\mathbf{n}\}|}; \quad n_y = \frac{\left(\frac{\partial z}{\partial r} \frac{\partial x}{\partial s} - \frac{\partial x}{\partial r} \frac{\partial z}{\partial s}\right)}{|\{\mathbf{n}\}|}; \quad n_z = \frac{\left(\frac{\partial x}{\partial r} \frac{\partial y}{\partial s} - \frac{\partial y}{\partial r} \frac{\partial x}{\partial s}\right)}{|\{\mathbf{n}\}|};$$

$$dS = \sqrt{\left(\frac{\partial y}{\partial r} \frac{\partial z}{\partial s} - \frac{\partial z}{\partial r} \frac{\partial y}{\partial s}\right)^2 + \left(\frac{\partial z}{\partial r} \frac{\partial x}{\partial s} - \frac{\partial x}{\partial r} \frac{\partial z}{\partial s}\right)^2 + \left(\frac{\partial x}{\partial r} \frac{\partial y}{\partial s} - \frac{\partial y}{\partial r} \frac{\partial x}{\partial s}\right)^2} dr ds. \quad (10.60)$$

At last, the common equations system of a dynamic balance of a shell of an arbitrary contour is as follows:

$$[\mathbf{M}]\{\ddot{\mathbf{U}}\} + [\mathbf{C}]\{\dot{\mathbf{U}}\} + [\mathbf{K}]\{\mathbf{U}\} = \{\mathbf{F}(t)\} \quad (10.61)$$

### 10.3.2. Input data for the solution of the problem

To solve the problem of the shells theory at dynamic loadings, the *Dynamic\_shell* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary *qualifier* of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of groups of finite elements (*ngr*) and the number of the known boundary conditions (*nbond*) are coded. Those elements, which have identical values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$  of a *damping*, which define a *damping matrix* are included into the corresponding group of the finite elements.

In the *second* line, the number of nodes, where additional coefficients of stiffness and dampings (*nst*) are given; the number of degrees of freedom in the direction of which external forces affect (*nforc*), and the number of finite elements, onto which the distributed loadings (*nq*) are acted, are coded. In the *third* line, a print code of intermediate results (*kprint*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. If parameter *ksolve*=0, then to solve, the `\solve` function of the Maple-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In the *fourth* line, the number of integration steps (*ntime*), and an integration step (*dtime*) are coded. In the *fifth* line, the number of eigenvalues (*neigen*); the number of iterations (*nitero*); precision of definition of eigenvalues and vectors (*tolerv*), are coded. If definitions of eigenvalues and vectors are not required, then relation *neigen*=0 is supposed. In the *sixth* line, the number of degrees of freedom, for which the values of results are recorded (*nout*) into the file; and the number indicating through how much integration steps the results of evaluations are recorded (*nstep*), are coded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*,*nnode*), where: *nnode* – number of nodes of the finite element, i.e. *nnode*=9} are coded, and also the number of the group, to which this finite element {array **Mgr**(*nelem*)} belongs. After arrays **Mtop** and **Mgr** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into each line of **Lbond** array the numbers of nodes, in which boundary conditions are known, are coded.

After array **Lbond**, the elements of **Lst**(*nst*) array are coded line by line. Into each line of the file the line number and an element of **Lst** array are recorded. Into each line of **Lst** array the number of a node, in which additional coefficients of stiffness and dampings are given, are recorded. After array **Lst**, the elements of **Lforc**(*nforc*) array are coded line by line. Into each line of the file a line number and the element of **Lforc** array are recorded. Into each line of **Lforc** array the numbers of degrees of freedom, in the direction of which external forces affect, are recorded. After array **Lforc**, the elements of **Lq**(*nq*) array are coded line by line. Into each line of the file a line number and the element of **Lq** array are recorded. Into each line of **Lq** array the number of a finite element, onto which the distributed loading affects, is recorded.

After array **Lq** the elements of **Lout**(*nout*) array are coded line by line. Into each line of the file a line number and the element of **Lout** array are recorded. Into the **Lout** array, the numbers of degrees of freedom, for which the values of results are recorded into the file, are coded. Behind array **Lout** the elements of arrays **Coord**(*npoin*, *ndime*) and **Coordk**(*npoin*, *ndime*), where *ndime* – dimensionality of the problem (*ndime*=3), are recorded line by line into the data file. The *x*-coordinates are recorded into the *first* column of the **Coord** array, the *y*-coordinates into the *second*, and *z*-coordinates of nodes of finite elements are recorded into the *third*; while the *x*, *y*, *z* coordinates of points, being on the surface of the shell and corresponding to nodes of a finite element are recorded into 4<sup>th</sup>, 5<sup>th</sup> and 6<sup>th</sup>. The number of a node, its (*x*, *y*, *z*)-coordinates and also (*x*, *y*, *z*)-coordinate of points, being on a surface of the shell, are recorded into each line of the file.

Behind arrays **Coord** and **Coordk**, the elements of **Bond**(*nbond*) array are recorded line by line. A line number of **Bond** array and a value of boundary condition in a node are recorded into each line of the datafile. The lines of **Bond** array should correspond to the lines of **Lbond** array. Behind **Bond** array, the elements of **Pgr**(*ngr*, 5) array are coded line by line. A line number of **Pgr** array, values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$ , which define a *damping matrix*, are coded into each line of the datafile. The line number of **Pgr** array

should strictly correspond to the number of group of finite elements.

After array **Pgr** the elements of **Ast**(*nst*, 10) array are recorded line by line. A line number of **Ast** array, five values of additional coefficients of stiffness, and five values of additional coefficients of damping, which correspond to each degree of freedom in a node, are recorded into each line of the datafile. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After array **Ast**, the elements of **Forc**(*nforc*, 4) array are recorded line by line. A line number of **Forc** array and values of coefficients  $\mathbf{a}_i$  ( $i=0..3$ ), which define the concentrated  $\mathbf{f}_k(\mathbf{t})$ -forces  $\mathbf{f}_k(\mathbf{t}) = [\mathbf{a}_0 + \mathbf{a}_1 \sin(\mathbf{a}_2 \mathbf{t})] e^{\mathbf{a}_3 \mathbf{t}}$  ( $k=1..3$ ), acting in the direction of  $k$ -th degree of freedom in the corresponding node are recorded into each line. The lines of **Forc** array should correspond to the lines of **Lforc** array.

After array **Forc**, the elements of **Fq**(*nq*, 13) arrays are recorded line by line. A line number of array **Fq** and the values of coefficients  $\mathbf{q}_1, \dots, \mathbf{q}_9, \mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$  are recorded into each line, by which the surface  $\mathbf{f}_s(\mathbf{r}, \mathbf{s}, \mathbf{t})$ -forces are defined being computed as follows:

$$\mathbf{f}_s(\mathbf{r}, \mathbf{s}, \mathbf{t}) = \sum_{i=1}^9 \mathbf{q}_i \mathbf{N}_i(\mathbf{r}, \mathbf{s}) [\mathbf{b}_0 + \mathbf{b}_1 \sin(\mathbf{b}_2 \mathbf{t})] e^{\mathbf{b}_3 \mathbf{t}} \quad (\mathbf{q}_i - \text{values of surface forces in nodes of a finite element}),$$

are recorded. The lines of array **Fq** should strictly correspond to the lines of array **Lq**. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. When reading the information from the datafile the following text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nst, nforc, nq, kprint, ksolve, niter, toler, ntime, dtime, neigen, nitero, tolero, nout, nstep*  
 Text line \*  
 Arrays **Mtop**(*nelem, nnode*), **Mgr**(*nelem*)  
 Text line \*  
 Array **Lbond**(*nbond*)  
 Text line \*  
 Array **Lst**(*nst*)  
 Text line \*  
 Array **Lforc**(*nforc*)  
 Text line \*  
 Array **Lq**(*nq*)  
 Text line \*  
 Arrays **Coord**(*npoin, ndime*), **Coordk**(*npoin, ndime*)  
 Text line \*  
 Array **Bond**(*nbond*)  
 Text line \*  
 Array **Pgr**(*ngr, 5*)  
 Text line \*  
 Array **Ast**(*nst, 10*)  
 Text line \*  
 Array **Forc**(*nforc, 4*)  
 Text line \*  
 Array **Fq**(*nq, 13*)

### 10.3.3. Brief description of the *Dynamic\_shell* program solving the problem

The *Dynamic\_shell* program was programmed on the *Maple*-language; it consists of the basic program and 70 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and time of its solution depend on the used number of finite elements and the number of nodes. The program calculates values of *displacements*, *velocities* and *accelerations* of nodes of finite elements in time dependence and *natural frequencies* and *forms*. The calculation results are output on the monitor and are recorded into a file; therefore *file\_rez1* of the program a qualifier of a target file must be ascribed to variable. *file\_rez1* values of *displacements*, *velocities* and *accelerations* of nodes of finite elements (*in a time dependence*) in degrees of freedom indicated in the array **Lout(nout)** and *natural frequencies* and *forms* are recorded into the datafile.

### 10.3.4. An example of use of the *Maple*-program *Dynamic\_shell*

As an example of application of the program, the motion of a plate made from *aluminium* is considered; the plate thickness is 0.005 m. (fig. 10.6).

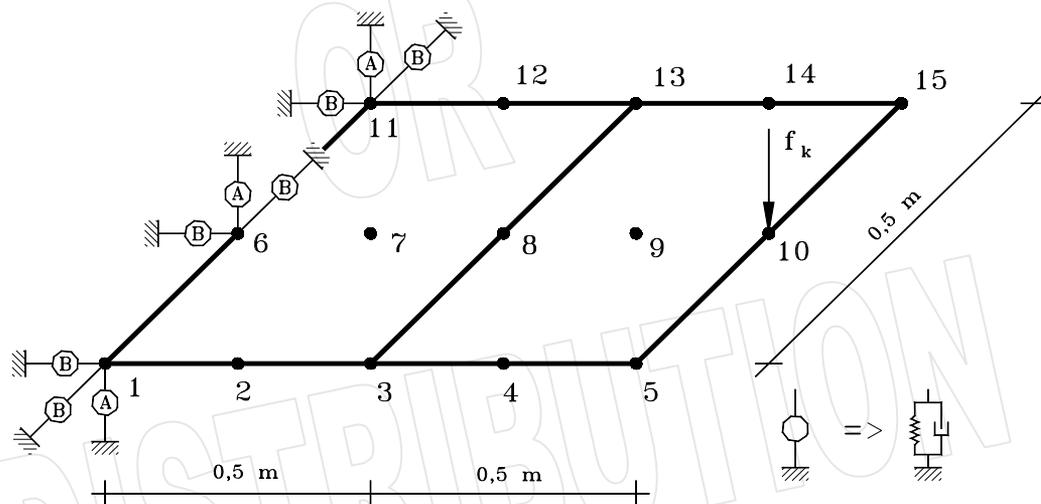


Fig. 10.6. The calculated scheme of a plate: an element **A** – additional coefficients of stiffness and damping along an axis are introduced; an element **B** – the additional coefficients of damping along an axis and also around of an axis are introduced.

The following short-term concentrated  $f_k(t)$ -force affects onto the construction:

$$f_k(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}, \text{ where } a_0 = -1H, a_1 = a_2 = 0, a_3 = -700 \text{ s}^{-1}.$$

Input data for the test example: number of finite elements  $nelem=2$ , the number of nodes  $npoin=15$ , the number of groups of finite elements  $ngr=1$ , the number of boundary conditions  $nbond=1$ ,  $kprint=0$ ,  $nq=1$ ,  $nst=3$ ,  $nforc=1$ ,  $ksolve=0$ ,  $niter=300$ ,  $toler=10^{-9}$ ,  $ntime=100$ ,  $dtime=10^{-4} \text{ s}$ ,  $neigen=5$ ,  $nitero=5$ ,  $nout=1$ ,  $nstep=1$ ,  $tolerv=10^{-3}$ .

Parameters of the first group:  $\rho = 2700 \frac{\text{kg}}{\text{m}^3}$ ;  $E = 70 \text{ GPa}$ ;  $\mu = 0,25$ ;  $\alpha_1 = \alpha_2 = 0$

Coefficients of additional stiffness and damping:  $k_1 = k_2 = k_3 = 10^{10} \frac{\text{N}}{\text{m}}$ ;  $k_4 = k_5 = 10^{10} \text{ N}\cdot\text{m}$ ;

$$\xi_1 = \xi_2 = \xi_3 = 10^{-5} \frac{\text{N}\cdot\text{s}}{\text{m}}; \xi_4 = \xi_5 = 10^{-5} \text{ N}\cdot\text{m}\cdot\text{s}$$

Results of calculation over the test example:

Natural frequencies (Hz):

- 1 w=7.001667e-03
- 2 w=1.702919e-02
- 3 w=4.004386e-02
- 4 w=7.762316e-02
- 5 w=1.057130e-01

Natural forms:

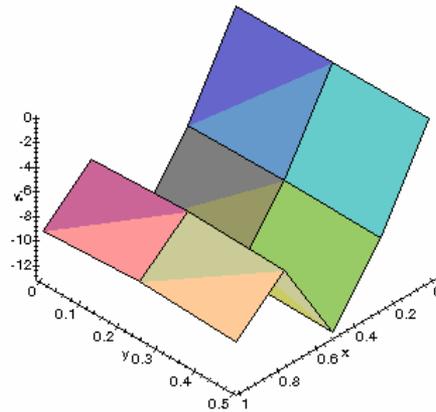
1	2.994859e-09	-4.019723e-10	-6.635346e-09	7.173787e-09	5.370754e-09
2	-1.432471e-10	-1.325096e-09	-3.508131e-10	-1.907001e-09	-6.902593e-10
3	5.660110e-11	2.243234e-10	1.919341e-10	4.491501e-10	3.219441e-11
4	-4.151265e-15	-1.553261e-14	-7.601604e-15	-1.800738e-14	-2.932290e-15
5	3.396915e-10	1.808864e-09	-2.747920e-10	-3.314724e-09	8.804168e-10
6	1.562558e-05	4.567874e-06	5.119480e-05	3.087469e-05	-6.812125e-05
7	1.931757e-02	1.170606e-02	-8.404081e-02	-5.049015e-02	6.403982e-02
8	-7.454419e+00	-5.652199e+00	6.214526e+00	-7.599441e+00	6.997509e+00
9	-3.333435e-05	-1.800519e-05	-5.206807e-06	-2.100276e-05	5.743348e-05
10	-1.612251e-03	-3.770486e-03	-3.804804e-02	-1.233657e-01	3.143607e-01
11	7.678541e-06	5.829128e-06	-3.553688e-06	1.054889e-04	-2.921257e-05
12	1.221849e-02	4.354746e-02	-1.441001e-01	-4.040861e-02	1.395638e-01
13	-1.285079e+01	-2.785702e+00	-1.461825e+01	-3.232342e+00	1.915118e+01
14	-1.161548e-05	-3.157809e-06	-2.176766e-06	-8.696283e-06	2.257465e-05
15	-2.617485e-03	-2.083556e-02	-8.295218e-03	-2.713528e-01	4.563084e-01
16	-1.590591e-07	5.904004e-06	-5.032114e-05	1.511151e-04	-1.683957e-06
17	-2.436701e-02	4.745294e-02	-9.710190e-02	-8.298973e-02	7.058963e-02
18	-5.563584e+00	7.091381e+00	-1.248527e+00	-1.646297e+01	8.752467e+00
19	-1.279210e-05	-2.787308e-06	1.676545e-06	-8.710008e-06	2.793996e-05
20	-5.012610e-03	-2.319122e-02	-1.054781e-01	-3.508155e-01	4.077089e-01
21	7.439230e-06	7.881409e-06	-2.299847e-05	1.429078e-04	-4.947391e-05
22	-1.136016e-02	7.002020e-02	-2.600437e-02	-6.155677e-02	1.862678e-02
23	-9.126731e+00	-3.261271e-01	3.357763e+00	4.863877e-01	2.288459e+01
24	-1.592906e-05	-6.780209e-06	-8.633179e-07	-8.924635e-06	3.104043e-05
25	-8.113000e-03	-3.184913e-02	-2.050091e-01	-3.827014e-01	1.118528e-01
26	-1.242353e-09	6.588907e-10	-4.171432e-09	3.565828e-09	7.367349e-10
27	-1.020444e-09	1.858296e-10	-4.069245e-09	3.879454e-09	4.838972e-09
28	-3.377193e-12	-1.779635e-10	-5.718727e-11	-2.169457e-10	-7.757955e-11
29	7.691708e-16	2.008209e-15	1.290792e-15	1.944970e-15	-1.945931e-16
30	2.054627e-09	1.153761e-08	-1.812289e-09	-2.342534e-08	3.744262e-09

31	-2.858719e-06	-7.907587e-07	-1.219658e-05	5.936760e-06	1.093390e-05
32	-1.071810e-02	1.589989e-02	8.191560e-03	4.007469e-02	4.560527e-02
33	-7.372027e+00	-5.461610e+00	8.123970e+00	-1.447587e+00	-8.717636e+00
34	7.011046e-05	4.272976e-05	1.205511e-05	3.980789e-05	-1.135926e-04
35	-1.779658e-05	-7.845973e-05	-1.018240e-05	2.886311e-04	-6.156657e-05
36	5.570906e-07	-4.594071e-07	2.222093e-06	-2.879450e-06	-4.937435e-06
37	-1.547816e-02	2.371148e-02	1.575205e-02	6.590357e-02	8.310959e-02
38	-1.271854e+01	-1.732142e+00	-1.425779e+01	1.037055e+01	-3.635048e+00
39	2.275772e-05	3.526119e-06	5.463736e-06	1.757057e-05	-4.502884e-05
40	1.749782e-05	2.279738e-04	1.873788e-04	-9.778328e-04	1.262555e-05
41	2.276672e-06	3.973077e-07	7.689950e-06	-6.597557e-06	-9.308053e-06
42	-1.092509e-02	3.346200e-02	7.844373e-03	1.051584e-01	1.402035e-01
43	-5.309628e+00	8.256136e+00	4.022524e+00	1.058363e+00	-1.162847e+01
44	2.986371e-05	3.347438e-06	-6.567660e-06	2.050292e-05	-6.897615e-05
45	-1.607910e-05	-6.484725e-05	1.416441e-05	2.172815e-04	-7.758879e-05
46	-7.206630e-07	6.419160e-07	-7.474013e-06	2.368655e-06	9.074755e-06
47	-1.795957e-02	3.344170e-02	-2.611642e-03	1.177061e-01	1.745164e-01
48	-8.726058e+00	1.259094e+00	1.364488e+01	1.963215e+01	1.730565e+01
49	-4.911893e-05	-2.381661e-05	-4.268533e-06	-3.495626e-05	8.500807e-05
50	-3.214860e-05	-9.241099e-05	-5.187874e-05	3.804043e-04	-1.324319e-05
51	3.877203e-09	-2.407459e-10	-2.647572e-09	2.018984e-10	1.335705e-09
52	7.355954e-11	-1.303867e-09	6.261686e-11	-2.262154e-09	-9.857178e-10
53	1.496045e-10	7.449756e-10	2.755967e-10	6.310606e-10	2.020039e-10
54	-9.087138e-16	-1.391354e-15	-3.065562e-15	6.270128e-16	2.170063e-15
55	3.213456e-10	1.470434e-09	-3.723857e-10	-3.684360e-09	3.926007e-10
56	4.456276e-06	1.385115e-06	3.073714e-05	-7.719633e-05	-8.232566e-06
57	6.395175e-03	1.275006e-03	5.705880e-02	-1.472639e-01	-2.088232e-01
58	-7.454429e+00	-5.653321e+00	6.215259e+00	-7.600412e+00	6.996847e+00
59	-3.333435e-05	-1.800517e-05	-5.206822e-06	-2.100271e-05	5.743350e-05
60	1.684514e-03	3.906122e-03	3.827393e-02	1.227085e-01	-3.141361e-01
61	-1.019957e-05	-2.234621e-06	-1.681231e-05	-8.696779e-05	6.836985e-05
62	4.805724e-03	3.001177e-02	9.559244e-02	-2.796469e-01	-3.909932e-01
63	-1.285071e+01	-2.787247e+00	-1.461611e+01	-3.235499e+00	1.914984e+01
64	-1.161548e-05	-3.157801e-06	-2.176769e-06	-8.696267e-06	2.257466e-05
65	2.687151e-03	2.123572e-02	6.130305e-03	2.728275e-01	-4.553907e-01
66	-1.417752e-05	-7.793335e-06	-1.426007e-05	-9.926878e-05	8.716983e-05
67	-1.486729e-02	4.105137e-02	1.297672e-01	-4.828432e-01	-6.122782e-01
68	-5.563583e+00	7.089762e+00	-1.247304e+00	-1.646461e+01	8.751450e+00

69	-1.279211e-05	-2.787334e-06	1.676565e-06	-8.710072e-06	2.793995e-05
70	5.113443e-03	2.341875e-02	1.054485e-01	3.502112e-01	-4.072855e-01
71	-2.110638e-06	-1.111558e-05	5.432369e-05	-1.422417e-04	3.097658e-05
72	5.495218e-03	7.089314e-02	1.533431e-01	-5.872168e-01	-7.360494e-01
73	-9.126758e+00	-3.260610e-01	3.357374e+00	4.869597e-01	2.288476e+01
74	-1.592906e-05	-6.780227e-06	-8.633152e-07	-8.924661e-06	3.104042e-05
75	8.118493e-03	3.178911e-02	2.056726e-01	3.820372e-01	-1.120771e-01

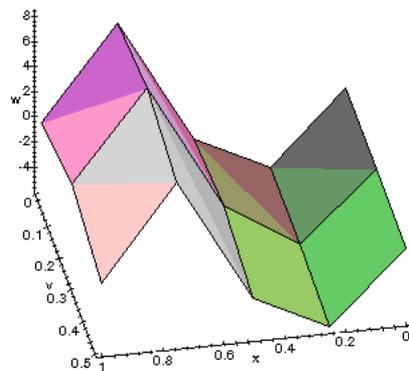
On fig. 10.7 natural forms of the explored plate are represented.

Natural Frequency Mode :



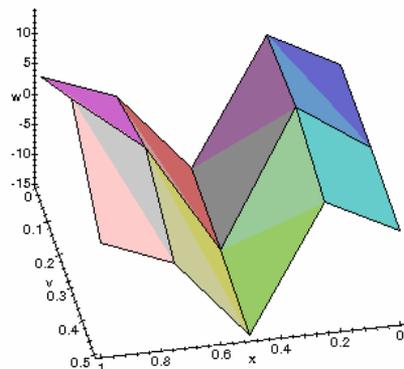
(a)

Natural Frequency Mode :



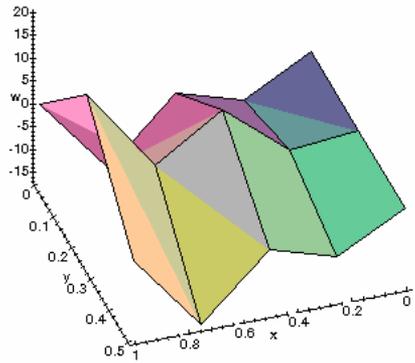
(b)

Natural Frequency Mode :



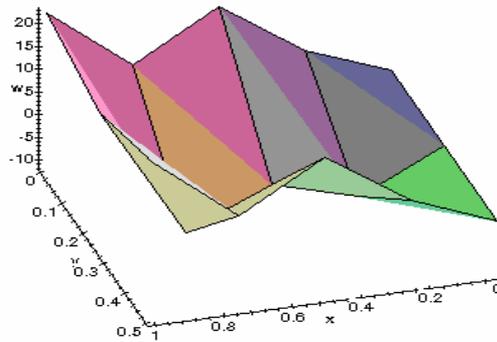
(c)

Natural Frequency Mode :



(d)

Natural Frequency Mode :

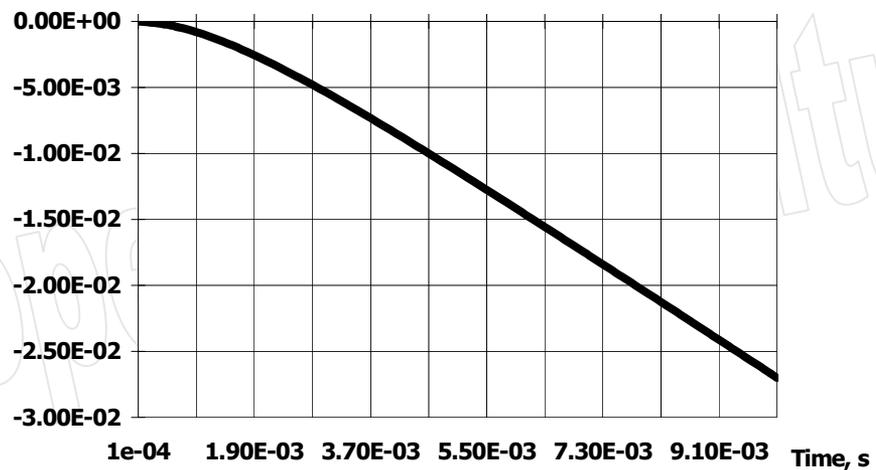


(e)

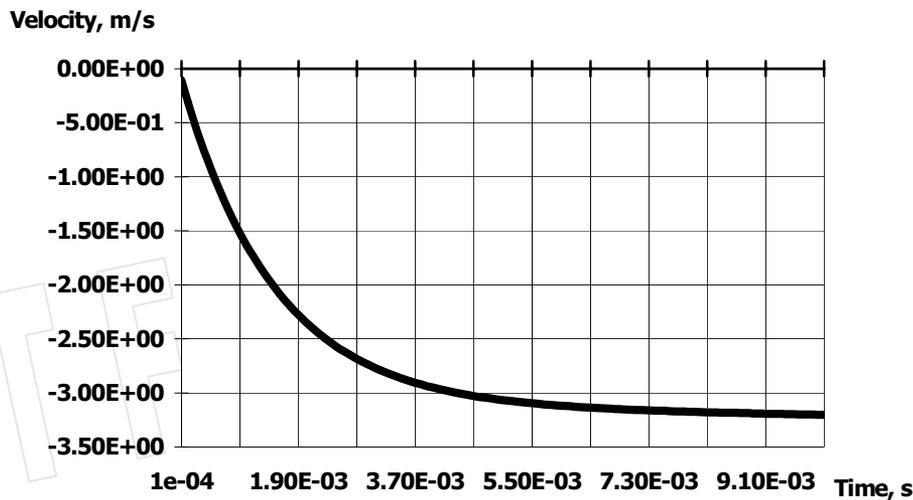
Fig. 10.7. The plate forms: (a) – the first form; (b) – the second form; (c) – the third form; (d) – the fourth form; (e) – the fifth form

On fig. 10.8 the dependences of *vertical displacements* (a) and *velocities* (b) of a node with the number 10 of a finite element in a time are represented.

Displacement, m



(a)



(b)

Fig. 10.8. The dependences of vertical displacements (a) and velocities (b) of a node with the number 10 of a finite element in a time

The *Dynamic\_shell* program is intended for the solution of a problem of the shells theory at dynamic loadings. Natural frequencies and forms are determined, and the system of dynamic balance is integrated in time. The source module of the program in Maple-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles Mb7\_3\_new.mws, Mb7\_3.dat and Mb7\_3\_1.rez accordingly.

### 10.4. Geometrically nonlinear problem of the theory of elasticity at dynamic loadings

As it is well known, the objects of mechanical engineering, the building and other constructions under action of external loadings are deformed. At such deformations the displacements of objects can have a considerable magnitude, and their tense state is described by the Hooke law. Such problems of the theory of elasticity are named as *geometrically nonlinear problems*. The nonlinearity occurs in expressions between a vector of strains and a vector of displacements. Owing to the given nonlinearity the obtained equations of motion of deformable bodies are *nonlinear equations*. In the present section, a dynamic problem of the theory of elasticity is considered in the presence of a nonlinear dependence between a vector of strains and a vector of displacements. The considered problem is solved by means of the above FEM.

#### 10.4.1. The calculated equations of the nonlinear problem

In this problem, the three-dimensional elastic body is considered, displacements of an arbitrary point which is defined by a vector given by the following relation:

$$\{U\}^T = [u(x, y, z, t) \quad v(x, y, z) \quad w(x, y, z, t)] \tag{10.62}$$

In the guess, the considered body had been made from an isotropic material, the dependence between stresses:

$$\{\sigma\}^T = [\sigma_{xx} \quad \sigma_{yy} \quad \sigma_{zz} \quad \sigma_{xy} \quad \sigma_{yz} \quad \sigma_{zx}] \tag{10.63}$$

and deformations:

$$\{\boldsymbol{\varepsilon}\}^T = [\varepsilon_{xx} \quad \varepsilon_{yy} \quad \varepsilon_{zz} \quad \varepsilon_{xy} \quad \varepsilon_{yz} \quad \varepsilon_{zx}] \quad (10.64)$$

acquires the following:

$$\{\boldsymbol{\sigma}\} = [\mathbf{D}]\{\boldsymbol{\varepsilon}\} \quad (10.65)$$

where  $[\mathbf{D}]$  – the following *elasticity matrix*:

$$[\mathbf{D}] = \frac{E}{(1+\mu)(1-2\mu)} \begin{bmatrix} 1-\mu & \mu & \mu & 0 & 0 & 0 \\ \mu & 1-\mu & \mu & 0 & 0 & 0 \\ \mu & \mu & 1-\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\mu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\mu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\mu}{2} \end{bmatrix}, \quad (10.66)$$

where  $E$  – the elastic modulus and  $\mu$  – the Poisson coefficient. In this case the dependence between *strains* and *displacements* is as follows:

$$\begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{yz} \\ \varepsilon_{zx} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial x} + \frac{1}{2} \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial x} \right)^2 + \left( \frac{\partial w}{\partial x} \right)^2 \right] \\ \frac{\partial v}{\partial y} + \frac{1}{2} \left[ \left( \frac{\partial u}{\partial y} \right)^2 + \left( \frac{\partial v}{\partial y} \right)^2 + \left( \frac{\partial w}{\partial y} \right)^2 \right] \\ \frac{\partial w}{\partial z} + \frac{1}{2} \left[ \left( \frac{\partial u}{\partial z} \right)^2 + \left( \frac{\partial v}{\partial z} \right)^2 + \left( \frac{\partial w}{\partial z} \right)^2 \right] \\ \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) + \left( \frac{\partial u}{\partial x} \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \frac{\partial v}{\partial y} + \frac{\partial w}{\partial y} \frac{\partial w}{\partial y} \right) \\ \left( \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \right) + \left( \frac{\partial u}{\partial z} \frac{\partial u}{\partial y} + \frac{\partial v}{\partial z} \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \frac{\partial w}{\partial y} \right) \\ \left( \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} \right) + \left( \frac{\partial u}{\partial x} \frac{\partial u}{\partial z} + \frac{\partial v}{\partial x} \frac{\partial v}{\partial z} + \frac{\partial w}{\partial x} \frac{\partial w}{\partial z} \right) \end{Bmatrix} \quad (10.67)$$

In the dependence (10.67) it is possible to pick out the *linear* and the *nonlinear* components, namely:

$$\varepsilon_i = \varepsilon_{L,i} + \varepsilon_{N,i}; \quad i=1..6 \quad (10.68)$$

Each component of the *strain vector* can be presented as follows [88, 89]:

$$\varepsilon_i = \{l_i\}^T \{g\} + \frac{1}{2} \{g\}^T [H_i] \{g\}, \quad (10.69)$$

where  $\{g\}$  – a vector of displacements gradients,

$$\{g\}^T = \left[ \frac{\partial u}{\partial x} \quad \frac{\partial v}{\partial x} \quad \frac{\partial w}{\partial x} \quad \frac{\partial u}{\partial y} \quad \frac{\partial v}{\partial y} \quad \frac{\partial w}{\partial y} \quad \frac{\partial u}{\partial z} \quad \frac{\partial v}{\partial z} \quad \frac{\partial w}{\partial z} \right]; \quad \{l_i\}, \quad (10.70)$$

$\{H_i\}$  - binary vectors and matrixes, for example:  $\{1_1\}^T = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$ ;  
 $[H_1] = \begin{bmatrix} [E] & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ ;  $\{1_5\}^T = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]$ ;  $[H_5] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & [E] \\ 0 & [E] & 0 \end{bmatrix}$ ;  $[E]$  - unit (3x3)-  
matrix. The *potential energy* of a deformable body is determined from the following relation:

$$\Pi = \Pi_D - W = \frac{1}{2} \int_V \{\sigma\}^T \{\varepsilon\} dV - \int_V \{u\}^T \{f_v\} dV - \int_S \{u\}^T \{f_s\} dS - \sum_i \{u\}_i^T \{f_k\}_i, \quad (10.71)$$

where  $\Pi_D$  - potential energy of a strain;  $W$  - a work of external forces;  $\{f_v\}$ ,  $\{f_s\}$ ,  $\{f_k\}$  - vectors of the volumetric, the surface and concentrated forces. By substituting expressions (10.65) and (10.69) into the expression for *potential energy of strain*, we obtain the following relation:

$$\Pi_D = \frac{1}{2} \int_V D_{ij} [\varepsilon_{L,i} \varepsilon_{L,j} + (\varepsilon_{L,i} \varepsilon_{N,j} + \varepsilon_{N,i} \varepsilon_{L,j}) + \varepsilon_{N,i} \varepsilon_{N,j}] dV \quad (10.72)$$

To solve the given problem the following *linear three-dimensional eight-nodal isoparametric finite element* is used (fig. 10.9):

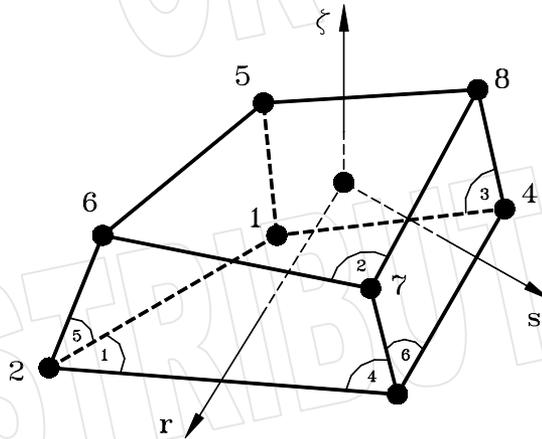


Fig. 10.9. Linear three-dimensional eight-nodal isoparametric finite element (in corners the numbers of surfaces of the finite element are indicated)

The *vector of displacements* of an arbitrary point of a finite element is approximated by the following relation:

$$\{u\} = [N]\{q\}, \quad (10.73)$$

where  $[N]$  - matrix of shape functions,

$$[N] = \begin{bmatrix} N_1 & 0 & 0 & N_2 & \dots & 0 & 0 & N_8 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & \dots & 0 & 0 & N_8 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & \dots & 0 & 0 & N_8 \end{bmatrix}; \quad (10.74)$$

$$N_i(r, s, \zeta) = \frac{1}{8} (1 + rr_i)(1 + ss_i)(1 + \zeta\zeta_i), \quad i=1..8; \quad (10.75)$$

$r_i$ ,  $s_i$ ,  $\zeta_i$  - the corresponding local coordinates of  $i$ -th node;  $\{u^{(e)}\}$  - vector of nodal displacements

of a finite element, and

$$\{\mathbf{u}^{(e)}\}^T = [u_1 \quad v_1 \quad w_1 \quad \dots \quad u_8 \quad v_8 \quad w_8]. \quad (10.76)$$

The *vector of displacements gradients* in a finite element is represented as follows:

$$\{\mathbf{g}\} = [\mathbf{G}]\{\mathbf{u}^{(e)}\}; \quad (10.77)$$

$$\text{where } \mathbf{G}_{ij} = \frac{\partial g_i}{\partial u_j}, \quad i=1..9, \quad j=1..24 \quad (10.78)$$

By substituting expressions (10.73) and (10.77) into the expression for potential energy of a body (10.71), we obtain the expression of potential energy for a finite element, namely:

$$\Pi^{(e)} = \frac{1}{2} \{\mathbf{u}^{(e)}\}^T \left[ \mathbf{k}_0^{(e)} \right] + \mathbf{k}_1^{(e)}(u) + \mathbf{k}_2^{(e)} \{\mathbf{u}^{(e)}\} - \{\mathbf{u}^{(e)}\}^T \{\mathbf{f}^{(e)}\} = \frac{1}{2} \{\mathbf{u}^{(e)}\}^T \left[ \mathbf{k}^{(e)} \right] \{\mathbf{u}^{(e)}\} - \{\mathbf{u}^{(e)}\}^T \{\mathbf{f}^{(e)}\}, \quad (10.79)$$

$$\text{where } \left[ \mathbf{k}_0^{(e)} \right] = \int_{V^{(e)}} [\mathbf{B}_L]^T [\mathbf{D}] [\mathbf{B}_L] \, dV; \quad (10.80)$$

$$\left[ \mathbf{k}_1^{(e)} \right] = 2 \int_{V^{(e)}} [\mathbf{B}_L]^T [\mathbf{D}] [\mathbf{B}_N] \, dV; \quad (10.81)$$

$$\left[ \mathbf{k}_2^{(e)} \right] = \int_{V^{(e)}} [\mathbf{B}_N]^T [\mathbf{D}] [\mathbf{B}_N] \, dV; \quad (10.82)$$

$$\mathbf{B}_{Lij} = \sum_{k=1}^9 \frac{\partial \epsilon_{Li}}{\partial g_k} \frac{\partial g_k}{\partial u_j}; \quad \mathbf{B}_{Nij} = \sum_{k=1}^9 \frac{\partial \epsilon_{Ni}}{\partial g_k} \frac{\partial g_k}{\partial u_j}; \quad (10.83)$$

$$\{\mathbf{f}^{(e)}\} = \int_{V^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_v\} \, dV + \int_{S^{(e)}} [\mathbf{N}]^T \{\mathbf{f}_s\} \, dS + \{\mathbf{f}_k\}. \quad (10.84)$$

The *kinetic energy* of a finite element can be presented as follows:

$$\mathbf{T}^{(e)} = \frac{1}{2} \int_{V^{(e)}} \rho \{\dot{\mathbf{u}}\}^T \{\dot{\mathbf{u}}\} \, dV = \frac{1}{2} \{\dot{\mathbf{u}}^{(e)}\}^T \left[ \mathbf{m}^{(e)} \right] \{\dot{\mathbf{u}}^{(e)}\}, \quad (10.85)$$

$$\text{where } \left[ \mathbf{m}^{(e)} \right] - \text{matrix of masses of a finite element and } \left[ \mathbf{m}^{(e)} \right] = \int_{V^{(e)}} \rho [\mathbf{N}]^T [\mathbf{N}] \, dV, \quad (10.86)$$

where  $\rho$  – density of a material. The *damping matrix* of a finite element can be presented in the following form:

$$\left[ \mathbf{c}^{(e)} \right] = \alpha_1 \left[ \mathbf{k}^{(e)} \right] + \alpha_2 \left[ \mathbf{m}^{(e)} \right]; \quad (10.87)$$

where  $\alpha_1, \alpha_2$  – known coefficients (*see section 10.1*). By using the Hamilton principle (*variation of the Lagrangian equals zero*):

$$\delta \int_{t_1}^{t_2} L dt = \delta \int_{t_1}^{t_2} (\Gamma^{(e)} - \Pi^{(e)}) dt = 0 \quad (10.88)$$

we obtain the *equations system* of dynamic balance for a finite element, namely:

$$[m^{(e)}]\{\ddot{u}^{(e)}\} + [c^{(e)}]\{\dot{u}^{(e)}\} + [k^{(e)}(u^{(e)})]\{u^{(e)}\} = \{f^{(e)}\} \quad (10.89)$$

Then *common equations system of a motion* of an elastic body is as follows:

$$[M]\{\ddot{U}\} + [C]\{\dot{U}\} + [K(U)]\{U\} = \{F(t)\} \quad (10.90)$$

The given equations system (10.90) is effectively solved by the method of trapezoids. The *vectors of displacements and velocities* at moment  $t + \tau$  are defined as follows:

$$\{U_{t+\tau}\} = \{U_t\} + \frac{\tau}{2} (\{\dot{U}_t\} + \{\dot{U}_{t+\tau}\}); \quad (10.91)$$

$$\{\dot{U}_{t+\tau}\} = \{\dot{U}_t\} + \frac{\tau}{2} (\{\ddot{U}_t\} + \{\ddot{U}_{t+\tau}\}), \quad (10.92)$$

where  $\tau$  - an integration step in a time. The *vector of accelerations* is determined from the expressions (10.91) and (10.92):

$$\{\ddot{U}_{t+\tau}\} = \frac{4}{\tau^2} (\{U_{t+\tau}\} - \{U_t\}) - \frac{4}{\tau} \{\dot{U}_t\} - \{\ddot{U}_t\} \quad (10.93)$$

The vector  $\{R\} = [K(U)]\{U\}$  at the point  $\{U\}_i$  is expanded into the Taylor series:

$$\{R\} = \{R\}_i + \frac{\partial}{\partial \{U\}_i^T} ([K(U)]\{U\}) \Big|_{\{U\}=\{U\}_i} \{\Delta U_i\} = \{R\}_i + [K_T(U_i)]\{\Delta U_i\}, \quad (10.94)$$

where  $[K_T(U_i)]$  - a tangential stiffness matrix. By using the method of trapezoids, we reduce a nonlinear equations system of a motion to the following form:

$$\left( \frac{4}{\tau^2} [M] + \frac{2}{\tau} [C] + [K_T] \right) \{\Delta U_{t+\tau}\}_i = \{F_{t+\tau}\}_i - [C]\{\dot{U}_{t+\tau}\}_i - [M]\{\ddot{U}_{t+\tau}\}_i - [K_{t+\tau}]_i \{U_{t+\tau}\}_i, \quad (10.95)$$

where  $i$  - number of iteration. When guesses are made, the *vectors of displacements, velocities and accelerations* after of the  $i$ -th iteration are determined by the following relations accordingly:

$$\{U_{t+\tau}\}_{i+1} = \{U_{t+\tau}\}_i + \{\Delta U_{t+\tau}\}_i; \quad (10.96)$$

$$\{\dot{U}_{t+\tau}\}_{i+1} = \{\dot{U}_t\}_i + \frac{\tau}{2} (\{\ddot{U}_{t+\tau}\}_i - \{\ddot{U}_t\}) + \frac{2}{\tau} \{\Delta U_{t+\tau}\}_i; \quad (10.97)$$

$$\{\ddot{U}_{t+\tau}\}_{i+1} = \frac{4}{\tau^2} \{\Delta U_{t+\tau}\}_i + \{\ddot{U}_{t+\tau}\}_i. \quad (10.98)$$

The iterative cycle of the solution of the nonlinear system of the algebraic equations is fulfilled on each integration step in time. The iterative cycle is terminated on reaching the given precision of the solution of the system, namely:

$$\max |\Delta U_{t+\tau}|_i < \varepsilon, \quad (10.99)$$

where  $\varepsilon$  - a precision of the solution.

### 10.4.2. Input data for the solution of the problem

To solve the *geometrically nonlinear problem* of the theory of elasticity at *dynamic loadings*, the *Dynamic\_body* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), the number of nodes (*npoin*), the number of groups of finite elements (*ngr*) and the number of the known boundary conditions (*nbond*) are coded. Into the corresponding group of the finite elements are included those elements, which have identical values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$  of a *damping*, which define a *damping matrix*.

In the *second* line, the number of nodes, in which the additional coefficients of stiffness and damping (*nst*) are given and number of degrees of freedom, in the direction of which the external forces affect (*nforc*), are coded. In the *third* line, a print code of intermediate results (*kprint*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*) are recorded. If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. If parameter *ksolve*=0, then to solve, the *'solve'* function, the *Maple-language* is used; otherwise, the equations system is solved by the method of *conjugate gradients*.

In the *fourth* line, the number of integration steps (*ntime*) and an integration step (*dtime*) are coded. In the *fifth* line, the number of iterations (*nitero*) and the precision of solution of nonlinear equations (*tol*) are coded. In the *sixth* line, the number of degrees of freedom, for which into the file the values of results are recorded (*nout*); and the number indicating through how much of integration steps the results of evaluations are recorded (*nstep*), are coded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop(nelem, nnode)**, where *nnode* - number of nodes of the finite element, i.e. *nnode*=8} are coded, and also the number of group, to which this finite element {array **Mgr(nelem)**} belongs. After arrays **Mtop** and **Mgr**, the elements of **Lbond(nbond)** array are coded line by line. The line number and element of **Lbond** array are recorded into each line of the file. The numbers of nodes, in which boundary conditions are known, are coded into each line of **Lbond** array.

After array **Lbond**, the elements of **Lst(nst)** array are coded line by line. The line number and an element of **Lst** array are recorded into each line of the file. The number of a node, in which additional coefficients of stiffness and dampings are given, are recorded into each line of **Lst** array. After array **Lst**, the elements of **Lforc(nforc)** array are coded line by line. A line number and the element of **Lforc** array are recorded into each line of the file. The numbers of degrees of freedom, in the direction of which the external forces affect, are recorded into each line of **Lforc** array.

After array **Lforc**, the elements of **Lout(nout)** array are coded line by line. A line number and the elements of **Lout** array are recorded into each line of the file. The numbers of degrees of freedom, for which the values of results are recorded into the file, are coded into the **Lout** array.

Behind array **Lout** the elements of **Coord(npoin, ndime)** array, where *ndime* - dimensionality of the problem (*ndime*=3), are recorded line by line into the data file. The *x*-coordinates are recorded into the *first* column of the **Coord** array, the *y*-coordinates are recorded into the *second* column of the **Coord** array; while the *z*-coordinates of nodes of finite elements are recorded into the *third*. The number of a node and its (*x, y, z*)-coordinates are recorded into each line of the file.

Behind array **Coord**, the elements of **Bond(nbond)** array are recorded line by line. A line number of **Bond** array and a value of boundary condition (*a value of displacement in a node*) are recorded into each line of the datafile. The lines of **Bond** array should correspond to the lines of **Lbond** array.

Behind **Bond** array the elements of **Pgr**(*ngr*, 5) array are coded line by line. A line number of **Pgr** array, values of *density* of a material, an *elastic modulus*, the *Poisson coefficient*, and *coefficients*  $\alpha_1$  and  $\alpha_2$  of a *damping*, which define a *damping matrix*, are recorded into each line of the datafile. The line number of the **Pgr** array should strictly correspond to the number of the group of finite elements.

After array **Pgr**, the elements of **Ast**(*nst*, 6) array are recorded line by line. A line number of **Ast** array, three values of additional coefficients of stiffness and three values of additional coefficients of damping, which correspond to each degree of freedom in a node, are recorded into each line of the datafile. The lines of **Ast** array should strictly correspond to the lines of **Lst** array.

After array **Ast**, the elements of **Forc**(*nforc*, 4) array are recorded line by line. Into each line, a line number of **Forc** array and values of coefficients  $\mathbf{a}_i$  ( $i=0..3$ ), which enter a definition of the concentrated forces  $\mathbf{f}_k(\mathbf{t}) = [\mathbf{a}_0 + \mathbf{a}_1 \sin(\mathbf{a}_2 \mathbf{t})] e^{\mathbf{a}_3 \mathbf{t}}$  ( $k=1..3$ ), acting in the direction of the  $k$ -th degree of freedom in the corresponding node, are recorded. The lines of **Forc** array should correspond to the lines of **Lforc** array. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. When reading information from the datafile, these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngr, nbond, nst, nforc, kprint, ksolve, niter, toler, ntime, dtime, niteration, toler, nout, nstep*  
 Text line \*  
 Arrays *Mtop(nelem, nmode), Mgr(nelem)*  
 Text line \*  
 Array *Lbond(nbond)*  
 Text line \*  
 Array *Lst(nst)*  
 Text line \*  
 Array *Lforc(nforc)*  
 Text line \*  
 Array *Coord(npoin, ndime)*  
 Text line \*  
 Array *Bond(nbond)*  
 Text line \*  
 Array *Pgr(ngr, 5)*  
 Text line \*  
 Array *Ast(nst, 6)*  
 Text line \*  
 Array *Forc(nforc, 4)*

**10.4.3. Brief description of the Dynamic\_body program solving the problem**

The *Dynamic\_body* program was programmed on the *Maple*-language; it consists of the basic program and 43 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and time of its solution depend on the used number of finite elements and number of nodes. The program calculates values of *displacements*, *velocities* and *accelerations* of nodes of finite elements in time dependence. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

Into the file *file\_rez1*, values of displacements, velocities and accelerations of nodes of finite elements (in a time dependence) in degrees of freedom indicated in the array **Lout(nout)** are recorded.

#### 10.4.4. An example of use of the Maple-program *Dynamic\_body*

As an example of application of the given program, the following problem of an arcuation of a three-dimensional elastic body, made from aluminium, is considered (fig. 10.10).

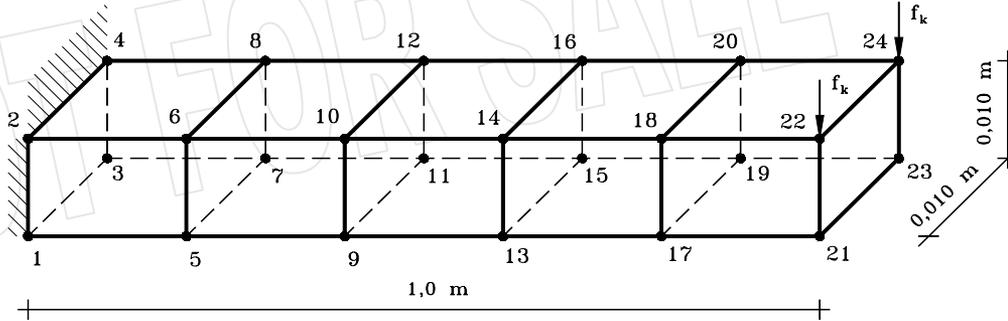


Fig. 10.10. The calculated scheme of the explored elastic body.

The following two concentrated  $f_k(t)$  forces affect onto the construction:

$$f_k(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}, \text{ where } a_0 = -100 \text{ N}, a_1 = a_2 = a_3 = 0.$$

Input data for the test example: number of the finite elements  $nelem=5$ , the number of nodes  $npoint=24$ , number of groups of finite elements  $ngr=1$ , the number of boundary conditions  $nbond=12$ ,  $kprint=0$ ,  $nst=1$ ,  $nforc=2$ ,  $ksolve=0$ ,  $niter=300$ ,  $toler=10^{-9}$ ,  $ntime=100$ ,  $dtime=10^{-4}$  s,  $nout=6$ ,  $nstep=1$ .

$$\text{Parameters of the first group: } \rho = 2700 \frac{\text{kg}}{\text{m}^3}; E = 70 \text{ GPa}; \mu = 0,25; \alpha_1 = \alpha_2 = 0$$

$$\text{Coefficients of additional stiffness and damping: } k_1 = k_2 = k_3 = 0; \xi_1 = \xi_2 = \xi_3 = 0$$

The *Dynamic\_body* program is intended for the solution of geometrically nonlinear problem of the theory of elasticity at dynamic loadings. The source module of the program in Maple-language, initial data for the test example, and also outcomes of its solution are represented in PROBLEMS directory of archive attached to the present book in datafiles *Mb7\_4\_new.mws*, *Mb7\_4.dat* and *Mb7\_4\_1.rez* accordingly.

# Chapter 11.

## Basic hydromechanics problems

The *hydromechanics* – science, which investigates the laws of a mechanical motion of fluids. The fluids over molecular structure occupy an intermediate position between gases and deformable solid bodies. In the hydromechanics a macroscopic motion of fluids and gases are considered, and also a force interaction of these mediums with solid bodies. The mathematical description of a motion of a fluid by *differential partial equations* which are taking into account all physical properties, is a rather composite problem. Moreover analytical solutions of such type of the equations, as a rule, are lacking. Therefore in the hydromechanics the various simplified models of medium and separate phenomena are widely used. One of such basic models is the model of an *incompressible ideal fluid* (or *non-viscous fluid*). Such model of a fluid gives a number of results confirmed by experience as *qualitatively*, and *numerically*; these results are useful for many important practical applications.

However there are cases of a motion of fluids, which cannot exactly be described without taking its *compressibility* into account. An example of such appearance is the *water hammer*. More fully the properties of an actual fluid are taken into account in a model of a *viscous fluid*, which represents a medium possessing *fluidity* and *viscosity*, but *absolutely incompressible*. The motion of a fluid can be *nonvortical*, or *potential* and *vortical*. At a nonvortical motion the components of a vector of velocity are partial derivatives of some function named as a *potential*. The structure of *vortical motions* of actual fluids is rather diversiform and intensively is explored.

### 11.1. Nonvortical motion of a fluids, described by potential of velocities

The *nonvortical* a fluid flow, which approximately simulates some actual currents, has major value in the hydromechanics. Such model allows with a sufficient degree of accuracy to solve many important engineering problems linked with flow around of bodies, with oscillations of buildings in a fluid and with a shock pulse action onto a fluid. In a singly-connected domain the *nonvortical fluid flow* is characterized by a single-valued function of coordinates and time – by a *potential of velocities* or by *pressure*.

#### 11.1.1. The calculated equations for potential motion of a fluid

*Potential of velocity* of a nonvortical motion is scalar function of radius of a vector of a point of space and time of a view  $\varphi = \varphi(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t})$ , the *gradient* of which equals a vector of velocity of a fluid, namely:

$$\bar{\mathbf{U}} = \text{grad}\varphi = \nabla\varphi \quad (11.1)$$

or in the projections onto an axes of a rectangular cartesian frame we obtain:

$$u_x = \frac{\partial\varphi}{\partial x}; \quad u_y = \frac{\partial\varphi}{\partial y}; \quad u_z = \frac{\partial\varphi}{\partial z} \quad (11.2)$$

In each point of space the vector of velocity of a fluid is orthogonal to potential surface, transiting

via this point. For a potential motion the fulfillment of the next *determinative relation* is necessary and sufficient condition [50]:

$$2\bar{\omega} = \nabla\bar{U} \quad (11.3)$$

for a *vortex vector*  $\bar{\omega}$  or the corresponding equalities:

$$\frac{\partial u_z}{\partial y} - \frac{\partial u_y}{\partial z} = 0, \quad \frac{\partial u_x}{\partial z} - \frac{\partial u_z}{\partial x} = 0, \quad \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} = 0 \quad (11.4)$$

for its components relative to cartesian axes.

According to the Lagrange's theorem follows: *if the vortex of velocity in all points of barotropic movable ideal fluid under activity of volumetric forces with single-valued potential in some initial moment was zero, then the motion is kept nonvortical and in any other subsequent moment of a time [50].* From a similar reasoning follows also, if in the beginning the motion was *vortical*, then it will remain the same and in the further. For potential current of a *compressible fluid* the *conservation equation of mass* accepts the following form:

$$\frac{\partial \rho}{\partial t} + \nabla(\rho\bar{U}) = 0 \quad (11.5)$$

A *motion equation* (the Euler's equation) at absence of activity of forces of masses accepts the following form:

$$\frac{\partial \bar{U}}{\partial t} + \frac{1}{2}\nabla(\bar{U}^2) + \frac{1}{\rho}\nabla p = 0 \quad (11.6)$$

The *equation of state* for a *barotropic process* of a motion can be written as follows:

$$\frac{p + A}{\rho_0 + A} = \left(\frac{\rho}{\rho_0}\right)^k, \quad (11.7)$$

where  $A = \frac{\rho_0 c_0^2}{k}$ ;  $c_0 = \sqrt{\frac{k\rho_0}{\rho_0}}$  - a velocity of a sound at a rest fluid and  $k$  - an isentropic exponent.

By substituting the relation (11.1) into the motion equation (11.6), we shall receive the Cauchy-Lagrange's integral of the following form:

$$\frac{\partial \varphi}{\partial t} + \frac{1}{2}(\nabla\varphi)^2 + \int \frac{dp}{\rho} = C(t) \quad (11.8)$$

At motion of a fluid of a quiescence at  $\varphi(\mathbf{t} = 0) = 0$  and, using an equation of state  $p = p_0 \left(\frac{\rho}{\rho_0}\right)^k$ , we can determine the C-constant and to receive an expression for pressure, namely:

$$p = p_0 - \rho_0 \left[ \frac{\partial \varphi}{\partial t} + \frac{1}{2}(\nabla\varphi)^2 - \frac{1}{2c_0^2} \left( \frac{\partial \varphi}{\partial t} + \frac{1}{2}(\nabla\varphi)^2 \right)^2 \right] \quad (11.9)$$

The *basic equation* of potential current of a *compressible fluid* has the following form:

$$\nabla^2 \varphi = \frac{1}{c_0^2} \left[ \frac{\partial^2 \varphi}{\partial t^2} + (k-1) \frac{\partial \varphi}{\partial t} \nabla^2 \varphi + 2\nabla\varphi \frac{\partial}{\partial t} (\nabla\varphi) \right], \quad (11.10)$$

where  $\nabla^2 \varphi$  - the Laplace's operator. In Cartesian, cylindrical ( $\mathbf{x} = r\cos\theta$ ,  $\mathbf{y} = r\sin\theta$ ,  $\mathbf{z} = z$ ) and spherical ( $\mathbf{x} = r\sin\theta\cos\vartheta$ ,  $\mathbf{y} = r\sin\theta\sin\vartheta$ ,  $\mathbf{z} = r\cos\vartheta$ ) coordinates the Laplace's operator accepts accordingly the following form:

$$\nabla^2 \varphi = \frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} + \frac{\partial^2 \varphi}{\partial z^2}; \tag{11.11}$$

$$\nabla^2 \varphi = \frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial \varphi}{\partial r} \right) + \frac{1}{r^2} \frac{\partial^2 \varphi}{\partial \theta^2} + \frac{\partial^2 \varphi}{\partial z^2}; \tag{11.12}$$

$$\nabla^2 \varphi = \frac{1}{r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial \varphi}{\partial r} \right) + \frac{1}{r^2 \sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial \varphi}{\partial \theta} + \frac{1}{r^2 \sin^2 \theta} \frac{\partial^2 \varphi}{\partial \theta^2} \right) \tag{11.13}$$

For solution of the equation (11.10) it is necessary to define the *initial conditions*:  $\varphi(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t} = 0) = \varphi_0$ ,  $\frac{d\varphi}{dt}(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{t} = 0) = \dot{\varphi}_0$  and *boundary conditions* of Dirichlet, Neumann or mixed type. If the non-stationary motion of an ideal fluid is conditioned by small perturbations of thermodynamic parameters, then the equation (11.10) turns into a *linear wave equation* of the following form:

$$\nabla^2 \varphi = \frac{1}{c_0^2} \frac{\partial^2 \varphi}{\partial t^2} \tag{11.14}$$

For an *incompressible fluid*  $c_0 = \infty$  the equation (11.14) turns into the Laplace's equation, namely:

$$\nabla^2 \varphi = 0 \tag{11.15}$$

while the *hydrodynamic pressure* is defined by the following dependence:

$$\mathbf{p} = \mathbf{p}_0 - \rho_0 \left[ \frac{\partial \varphi}{\partial t} + \frac{1}{2} (\nabla \varphi)^2 \right] \tag{11.16}$$

The investigation of a *stationary motion of a fluid* is restricted by the solution of a boundary-value problem for the Laplace's equation. The two-dimensional Laplace equation (11.10) is solved by means of the FEM, applying the Bubnov-Galerkin's procedure. *Potential of velocity* in limits of a finite element is approximated by the next expression [70]:

$$\varphi(\mathbf{r}, \mathbf{s}) = \sum_{i=1}^n \mathbf{N}_i(\mathbf{r}, \mathbf{s}) \varphi_i = [\mathbf{N}]\{\varphi\}, \tag{11.17}$$

where  $[\mathbf{N}(\mathbf{r}, \mathbf{s})] = [\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_n]$  - matrix of shape functions and  $\{\varphi\}$  - vector of nodal generalized coordinates of an element. The *quadrangular isoparametric finite elements*, i.e.  $n=4$ , are used (fig. 11.1).

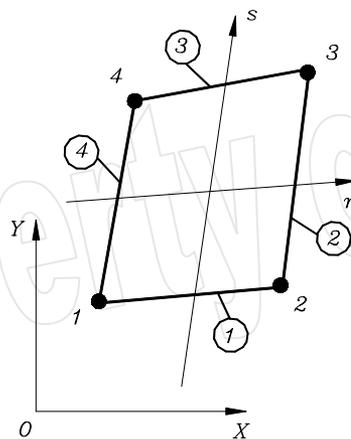


Fig. 11.1. A quadrangular isoparametric finite element (the number in a circle indicates the number of an element side)

By substituting expression (11.17) into the Laplace's equation (11.15) and by applying the Bubnov-Galerkin's procedure, we obtain the following relation:

$$\int_A [N]^T \nabla^2 ([N]\{\phi\}) dA = 0 \quad (11.18)$$

By integrating the obtained expression in parts, we shall receive the following matrix equation for definition of a finite element:

$$[k^{(e)}]\{\phi\} = \{f^{(e)}\}, \quad (11.19)$$

where  $[k^{(e)}] = \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) x^\gamma dA;$  (11.20)

$$\{f^{(e)}\} = \int_{\Gamma^{(e)}} [N]^T x^\gamma \left( \frac{\partial \phi}{\partial x} I_x + \frac{\partial \phi}{\partial y} I_y \right) d\Gamma; \quad (11.21)$$

$I_x, I_y$  - direction cosines;  $\gamma=0$ , using cartesian coordinates, and  $\gamma=1$ , using cylindrical coordinates at axial-symmetric motion ( $x=r, y=z$ ). For solution of the given problem the following boundary conditions are used:

- the Dirichlet's condition - on a contour  $\Gamma_\phi$  a value of potential of velocity is known:

$$\phi = \phi_0 \quad (11.22)$$

- the Neumann's condition - the velocity of a motion of a fluid  $u_n$  over a normal line to a contour  $\Gamma_u$  is known (fig. 11.2):

$$u_n = \frac{\partial \phi}{\partial n} = \frac{\partial \phi}{\partial x} I_x + \frac{\partial \phi}{\partial y} I_y \quad (11.23)$$

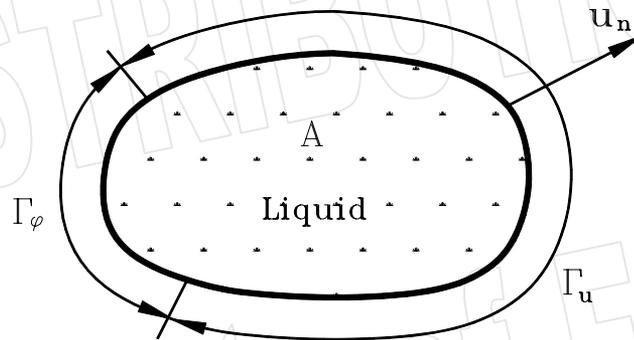


Fig. 11.2. Boundary conditions for solution of the Laplace's equation, written in variables of velocity potential

Derivatives of shape functions  $N_i(r, s)$  over global coordinates  $x$  and  $y$  are determined from the next expression:

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial r} \\ \frac{\partial N_i}{\partial s} \end{Bmatrix}, \quad (11.24)$$

where  $[J]$  - the Jacobi matrix,

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} = \begin{bmatrix} \sum_i^n \frac{\partial N_i}{\partial r} x_i & \sum_i^n \frac{\partial N_i}{\partial r} y_i \\ \sum_i^n \frac{\partial N_i}{\partial s} x_i & \sum_i^n \frac{\partial N_i}{\partial s} y_i \end{bmatrix}. \quad (11.25)$$

The elementary area of a finite element is defined by the following relation:

$$dA = x^y dx dy = x^y \det[J] dr ds \quad (11.26)$$

While an elementary length of a contour for sides 1, 3 and 2, 4 ( $ds=0$ ) of a finite element accepts the following form accordingly:

$$d\Gamma = \sqrt{\left(\frac{\partial x}{\partial r}\right)^2 + \left(\frac{\partial y}{\partial r}\right)^2} dr; \quad (11.27)$$

$$d\Gamma = \sqrt{\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2} ds \quad (11.28)$$

By taking into account now the relations (11.19), the *global equations system* of the following view is formed, namely:

$$[K]\{\Phi\} = \{F\} \quad (11.29)$$

### 11.1.2. Input data for the solution of the problem

For the solution of the Laplace's equation describing a stationary potential motion of a fluid, the *Pot\_flow* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary *qualifier* of the file is ascribed to variable F. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions (*nbond*) and the number of Neumann's boundary conditions (*ngrad*). In the *second* line, a type of a problem (*ngama*), a print code of intermediate results (*kprint*) and a density of fluid (*tankis*) are coded. If *kprint=0*, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama=0*; otherwise, in *cylindrical* frames an axial-symmetric problem is solved.

In the *third* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). If parameter *ksolve=0*, then for the solution, the *'solve'* function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where: *nnode* - number of nodes of the finite element, i.e. *nnode=4*} are coded. After of array **Mtop** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into the **Lbond** array the numbers of nodes, in which are known Dirichlet's boundary conditions, are coded.

Behind of array **Lbond** into the data file the elements of **Lgrad**(*ngrad*, 3) array are recorded line by line. Into each line of the file the line number and element of **Lgrad** array are recorded. Into the *first*

column of the **Lgrad** array the number of a finite element, while into the *second* and the *third* column the numbers of nodes of the element side on which the Neumann's boundary conditions are known, are recorded.

Behind of array **Lgrad** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, into the *second* column of the **Coord** array the *y*-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded. Behind of array **Coord** the elements of **Bond**(*nbond*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions, i.e. the values of *potential of velocity*, are recorded. Into each line of the datafile a line number of **Bond** array and a boundary value of *potential of velocity* are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

In each subsequent *ngrad* lines of the datafile, the elements of the **Grad**(*ngrad*, 2) array are recorded line by line. Into the *first* column of this array a value of derivative  $\frac{\partial\varphi}{\partial x}$ , and into the *second* column – a value of derivative  $\frac{\partial\varphi}{\partial y}$ , which are known on a contour of calculated area, are recorded (*the Neumann's boundary conditions*). Into each line of the datafile a line number of **Grad** array and values of partial derivatives  $\frac{\partial\varphi}{\partial x}$  and  $\frac{\partial\varphi}{\partial y}$  are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbond, ngrad, ngama, kprint, tankis, ksolve, niter, toler*  
 Text line \*  
 Arrays **Mtop**(*nelem, nmode*)  
 Text line \*  
 Array **Lbond**(*nbond*)  
 Text line \*  
 Array **Lgrad**(*ngrad, 3*)  
 Text line \*  
 Array **Coord**(*npoin, ndime*)  
 Text line \*  
 Array **Bond**(*nbond*)  
 Text line \*  
 Array **Grad**(*ngrad, 2*)

11.1.3. Brief description of the Pot\_flow program solving the problem

The *Pot\_flow* program was programmed on the Maple-language; it consists of the basic program and 31 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of *potential of velocity*, *projection of velocities* of a motion of a fluid onto axes *X* and *Y*, and also a *hydrodynamic pressure* in nodes of finite elements. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of *potential of velocity*, values

of velocities of a motion of a fluid in the direction of axes **X** and **Y**, and also a hydrodynamic pressure in nodes of finite elements are recorded. To variables *file\_geo* and *file\_dr* of the program also must be ascribed the qualifiers of files, in which the *F\_isograf* procedure saves an information for visualization of the image of *isolines*.

### 11.1.4. An example of use of the *Maple*-program *Pot\_flow*

The problem of flowing around of a sphere by an axial-symmetric fluid current in a cylindrical pipe of the round cross-section is considered (fig. 11.3). Contours of an explored object of a fluid are a spherical surface, a wall of a pipe and normal cross-sections of a pipe. An axial-symmetric current has a plane of symmetry passing via centre of a sphere, therefore it is quite enough to consider half of volume of a fluid.

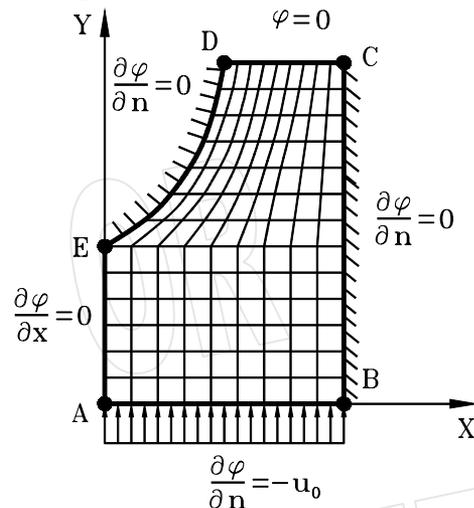


Fig. 11.3. Flowing around of a sphere by an axial-symmetric current in a pipe

At an *asymmetrical* motion of a fluid the potential of velocity does not depend on angular coordinate  $\theta$ . On the contour **CD** the boundary conditions of the Dirichlet  $\varphi = 0$  are given. On the contour **AB** the boundary conditions of the Neumann  $\frac{\partial \varphi}{\partial n} = -u_0 = 1 \text{ m/s}$  are given, and also the homogeneous conditions  $\frac{\partial \varphi}{\partial n} = 0$  on a sphere surface **DE**, on a wall **BC** and on a symmetry axis **EA** are accepted.

Input data for the test example: a volume of a fluid in a cylindrical pipe is divided onto *nelem*=117 finite elements, number of nodes *npoin*=140, number of boundary conditions of the Dirichlet *nbond*=10, number of boundary conditions of the Neumann *ngrad*=9, radius of a pipe 2 m; radius of a sphere 1 m; length of a pipe 3.5 m., *kprint*=0, *ksolve*=1, *ngama*=1, density of a fluid *tankis*= 1000 kg/m<sup>3</sup>.

#### Results of calculation over the test example:

The number of a node (**n**), potential of velocity (**pv**), velocity (**Vx**, **Vy**) in the direction of axes of coordinates **X**, **Y** accordingly, and pressure (**P**):

n	pv	Vx	Vy	P
node=1	3.751623	-.091226	.948587	454.069753
node=2	3.771895	-.050917	.991004	492.341395
node=3	3.774253	-.006019	.994342	494.376511
node=4	3.774571	.000554	.996455	496.461854

node=5	3.774007	.001783	.998468	498.470954
node=6	3.773778	.002448	.999771	499.774536
node=7	3.772919	.002708	1.000470	500.474135
node=8	3.772574	.001833	1.000817	500.819924
node=9	3.772104	.001561	1.001216	501.218396
node=10	3.771880	.001005	1.000661	500.662350
node=11	3.097420	.040417	.964421	465.871007
node=12	3.088439	.020083	.979923	480.327039
node=13	3.088495	.002438	.985351	485.462147
node=14	3.087355	.006956	.989448	489.528565
node=15	3.085403	.006928	.993014	493.063160
node=16	3.084276	.005555	.997643	497.661985
node=17	3.082934	.004332	1.001063	501.073275
node=18	3.082350	.002990	1.003739	503.751294
node=19	3.081605	.001318	1.005439	505.455665
node=20	3.081764	-.000716	1.005898	505.916134
node=21	2.421397	.004999	.941496	443.220430
node=22	2.420286	.014040	.934477	436.722535
node=23	2.415157	.023539	.945721	447.471519
node=24	2.409824	.024334	.961278	462.324042
node=25	2.404341	.027216	.978015	478.627807
node=26	2.397728	.027428	.993119	493.519259
node=27	2.392152	.021625	1.005616	505.866361
node=28	2.388116	.015413	1.014556	514.780844
node=29	2.385300	.008260	1.019543	519.768717
node=30	2.384445	.003851	1.021427	521.664452
node=31	1.954411	-.001113	.787991	310.466052
node=32	1.954658	.028407	.809039	327.675711
node=33	1.941785	.070099	.853445	366.641494
node=34	1.923503	.086474	.905771	413.950139
node=35	1.903352	.086122	.955718	460.407897
node=36	1.885227	.075277	.993133	495.990798
node=37	1.869898	.058550	1.020539	522.464537
node=38	1.859202	.038508	1.038297	539.772566
node=39	1.852780	.019050	1.047103	548.394519
node=40	1.850736	.009196	1.049786	551.068180
node=41	1.722276	.068276	.465320	110.592614
node=42	1.707103	.120390	.595194	184.375136
node=43	1.668769	.189153	.721925	278.477992
node=44	1.623034	.203741	.842626	375.764978
node=45	1.578216	.182414	.935939	454.628810
node=46	1.541963	.143966	1.003683	514.053762
node=47	1.514236	.104009	1.045569	552.016352

node=48	1.495733	.064323	1.069124	573.582172
node=49	1.485642	.029686	1.079926	583.560892
node=50	1.482540	.013959	1.084314	587.966837
node=51	1.677901	.235094	.009995	27.684747
node=52	1.625658	.273475	.364990	104.003395
node=53	1.556357	.321581	.607475	236.220427
node=54	1.482731	.311137	.791326	361.501834
node=55	1.418072	.259220	.929630	465.704105
node=56	1.367525	.194678	1.020824	539.991087
node=57	1.331554	.132045	1.070785	582.008670
node=58	1.308833	.078064	1.094133	601.611027
node=59	1.296852	.036154	1.105688	611.926803
node=60	1.292766	.018388	1.108849	614.942736
node=61	1.698365	.427093	.062851	93.179676
node=62	1.603456	.428697	.411615	176.604472
node=63	1.507835	.420741	.635388	290.370884
node=64	1.416458	.382401	.793486	387.925857
node=65	1.337876	.309842	.923993	474.883069
node=66	1.278754	.224080	1.020995	546.322091
node=67	1.238292	.146463	1.075880	589.484924
node=68	1.213655	.084730	1.106208	615.437970
node=69	1.200627	.038629	1.119032	626.862717
node=70	1.196488	.018626	1.123205	630.969022
node=71	1.507326	.515906	.216175	156.445672
node=72	1.414613	.496813	.510785	253.862859
node=73	1.328761	.439461	.775681	397.404515
node=74	1.256661	.352163	.961274	524.034149
node=75	1.202199	.258867	1.061505	596.903461
node=76	1.163612	.178042	1.107238	628.837908
node=77	1.138189	.115463	1.122578	636.757508
node=78	1.122114	.068364	1.131174	642.114512
node=79	1.113619	.032222	1.133924	643.411731
node=80	1.110534	.017170	1.134419	643.601546
node=81	1.370470	.638096	.535343	346.879765
node=82	1.267826	.577628	.770019	463.292116
node=83	1.184687	.450215	.959590	561.754107
node=84	1.123103	.329363	1.058476	614.426518
node=85	1.078865	.233279	1.106045	638.877885
node=86	1.048193	.159593	1.128353	649.325345
node=87	1.027648	.105057	1.142357	658.009311
node=88	1.014499	.063097	1.147224	660.052900
node=89	1.007419	.028833	1.152004	663.972920
node=90	1.005258	.013501	1.148919	660.098585

node=91	1.176337	.694200	.886129	633.570273
node=92	1.076115	.594548	.986498	663.333285
node=93	1.004756	.421398	1.092960	686.069712
node=94	.954644	.296066	1.142296	696.247961
node=95	.919523	.206161	1.163642	698.282854
node=96	.895369	.142098	1.169922	694.455772
node=97	.878719	.093385	1.172752	692.034753
node=98	.868593	.056121	1.170853	687.023742
node=99	.862639	.025621	1.172667	687.902984
node=100	.861258	.009665	1.172237	687.117093
node=101	.930212	.632917	1.213363	936.418073
node=102	.847806	.531389	1.193952	853.948941
node=103	.791881	.363881	1.221330	812.029344
node=104	.753154	.253151	1.224438	781.667809
node=105	.726062	.173931	1.219191	758.339621
node=106	.707960	.117303	1.212248	741.652870
node=107	.695610	.076345	1.205034	728.968352
node=108	.688152	.047656	1.199210	720.188744
node=109	.683257	.023983	1.196459	716.044880
node=110	.681936	.010201	1.196642	716.029172
node=111	.643503	.484994	1.462279	1186.741148
node=112	.585497	.403704	1.370670	1020.856875
node=113	.546937	270163	1.331975	923.573167
node=114	.520896	.182778	1.298026	859.139772
node=115	.503238	.125201	1.269280	813.374747
node=116	.490948	.087398	1.249819	784.843184
node=117	.482332	.057236	1.234406	763.517982
node=118	.477261	.035177	1.226980	753.359268
node=119	.473922	.017506	1.219235	743.420571
node=120	.473074	.007093	1.218443	742.327129
node=121	.329111	.265716	1.626661	1358.316501
node=122	.299006	.218755	1.489599	1133.381056
node=123	.279558	.144213	1.405435	998.023845
node=124	.266344	.097612	1.346872	911.796874
node=125	.257447	.068008	1.306247	855.453410
node=126	.250941	.045492	1.277700	817.293945
node=127	.247142	.030606	1.257163	790.698256
node=128	.244009	.019168	1.245167	775.404662
node=129	.242800	.008720	1.237064	765.202144
node=130	.242034	.006757	1.235077	762.731008
node=131	0	0	1.686887	1422.795186
node=132	0	0	1.532578	1174.398726
node=133	0	0	1.432900	1026.602271

node=134	0	0	1.365168	931.843117
node=135	0	0	1.319568	870.630449
node=136	0	0	1.286220	827.181418
node=137	0	0	1.266748	802.325470
node=138	0	0	1.250688	782.111473
node=139	0	0	1.244490	774.378093
node=140	0	0	1.240566	769.502416

The allocation of velocity  $u_y$  along boundary CD is shown on the fig. 11.4.

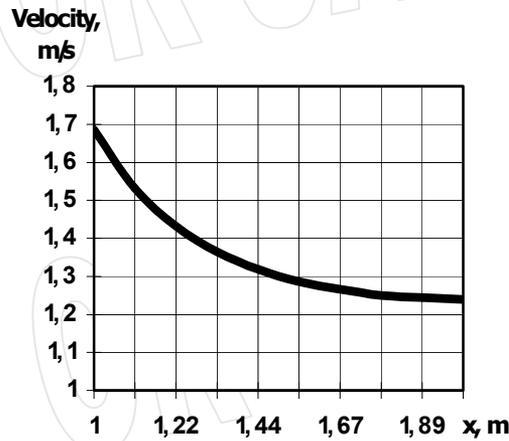


Fig. 11.4. The allocation of velocity  $u_y$  along boundary CD

While allocation of potential of velocity in volume of a fluid is represented by isolines on the fig. 11.5.

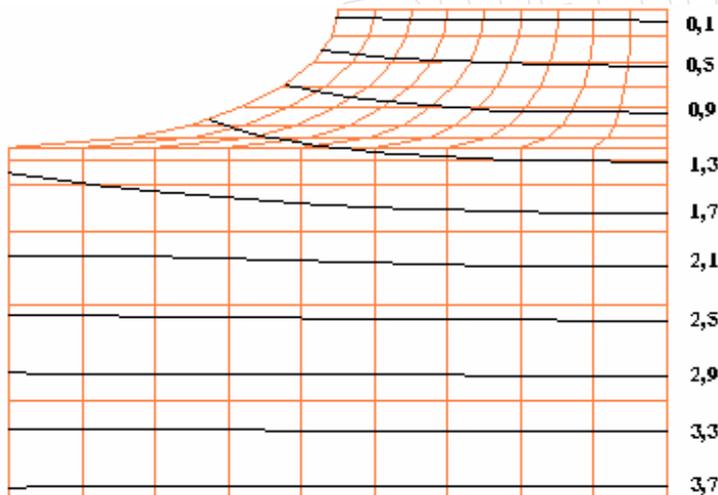


Fig. 11.5. Isolines of potential of velocity in volume of a fluid

The *Plot\_flow* program is intended for the solution of the stationary two-dimensional equation of Laplace written in cartesian or cylindrical axials for a case of an assymetrical problem. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in files *Mb4\_1\_new.mws*, *Mb4\_1.dat* and *Mb4\_1\_1.rez* accordingly.

## 11.2. Nonvortical motion of a fluid described by the stream function

The properties of currents, considered in the previous section, remain valid and for any motions of an *incompressible* or a *compressible* fluid. In the present section a partial case is considered, but being practically important case of a *flat current* of an incompressible fluid; i.e. in such current the configuration of streamlines in all planes, normal to some straight line, is identical. In a nature of flat currents of a fluid does not meet, however, there are very many of cases, when the stream with a particular degree of admissibility can be considered as a *flat*.

### 11.2.1. The calculated equations for motion of a fluid as the stream function

The *flat motion* is much easier for studying because: (1) equations describing the given motion are much easier, and (2) it is enough to explore a current only in one plane to make representation about the stream as a whole. The *continuity equation* for an *incompressible fluid* in the plane  $X, Y$  has the following form:

$$\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} = 0 \quad (11.30)$$

The *equation of streamlines* of a fluid can be presented as follows:

$$\frac{dx}{u_x} = \frac{dy}{u_y} \quad \text{or} \quad u_x dy - u_y dx = 0 \quad (11.31)$$

The *continuity equation* (11.30) is necessary and sufficient condition that the left-hand part of the expression (11.31) would be *differential* of some function of two variables. We shall denote this function as  $\psi$  and shall call it as a *stream function* [68]. Then we obtain:

$$d\psi = \frac{\partial \psi}{\partial x} dx + \frac{\partial \psi}{\partial y} dy = u_x dy - u_y dx \quad (11.32)$$

From here follows, that:

$$u_x = \frac{\partial \psi}{\partial y}; \quad \text{or} \quad u_y = -\frac{\partial \psi}{\partial x} \quad (11.33)$$

In mechanical engineering the especial value has an *axial-symmetric* motion, which is most convenient for describing in the cylindrical frame. At an axial-symmetric motion all its parameters do not depend on an angle  $\theta$ , and the *stream function* is defined by the following relations:

$$u_r = -\frac{1}{r} \frac{\partial \psi}{\partial z}; \quad u_z = \frac{1}{r} \frac{\partial \psi}{\partial z} \quad (11.34)$$

For any *streamline* takes place the relation (11.31), therefore *total differential* along the streamline will be equal zero, i.e.  $d\psi = 0$  or  $\psi(x, y) = \text{const}$ . Therefore, the stream function has property to maintain the constant value along any streamline, various for different streamlines. The difference of values of a stream function on two streamlines is equal to consumption of a fluid between them, i.e.:

$$Q = \int_{\Gamma} u_n d\Gamma = \int_{\Gamma} \bar{U}_n \bar{n} d\Gamma = \int_{\Gamma} (u_x l_x + u_y l_y) d\Gamma = \int_{\Gamma} \left( \frac{\partial \psi}{\partial y} dy + \frac{\partial \psi}{\partial x} dx \right) = \int_{\Gamma} d\psi = \psi_b - \psi_a \quad (11.35)$$

Then a *consumption of a fluid* via round cross-section of a stream of radius  $r$ , normal to axis  $Z$ , is defined by the following relation:

$$Q = 2\pi \int_{r_a}^{r_b} u_z r dr = 2\pi \int_{r_a}^{r_b} \frac{\partial \psi}{\partial r} dr = 2\pi[\psi(r_b, z) - \psi(r_a, z)] \quad (11.36)$$

The condition of a nonvortical motion in a plane accepts the following form:

$$\frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y} = 0 \quad (11.37)$$

By substituting expressions (11.33) into the condition (11.37), we obtain the equation of Laplace for a stream function of a fluid as follows:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = 0 \quad \text{or} \quad \nabla^2 \psi = 0 \quad (11.38)$$

The equation of Laplace written for a stream function in cylindrical axials, for an axial-symmetric motion accepts the following form:

$$\frac{1}{r} \frac{\partial}{\partial r} \left( r \frac{\partial \psi}{\partial r} \right) - \frac{2}{r} \frac{\partial \psi}{\partial r} + \frac{\partial^2 \psi}{\partial z^2} = 0 \quad (11.39)$$

For the solution of the equations (11.38) or (11.39) it is necessary to know boundary conditions of Dirichlet, Neumann or the *mixed type*. The two-dimensional equations of Laplace (11.38) or (11.39) is solved by means of the FEM, using the of Bubnov-Galerkin's procedure. The *stream function* in a finite element is approximated by expression of the following form:

$$\psi(r, s) = \sum_{i=1}^n N_i(r, s) \psi_i = [N]\{\psi\} \quad (11.40)$$

where  $[N(r, s)] = [N_1, N_2, \dots, N_n]$  - matrix of functions of the forms;  $\{\psi\}$  - vector of nodal generalized coordinates of an element (*value of stream functions of a fluid in nodes of an element*). We use quadrangular isoparametric finite elements, i.e.  $n=4$  (fig. 11.1). By applying the Bubnov-Galerkin's procedure for the equation of Laplace (11.38) or (11.39), we obtain the following relation:

$$\int_A [N]^T \left[ \frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma \frac{\partial \psi}{\partial x} \right) - \frac{2\gamma}{x} \left( \frac{\partial \psi}{\partial x} \right) + \frac{\partial^2 \psi}{\partial y^2} \right] dA = 0 \quad (11.41)$$

where  $\gamma=0$ , using cartesian coordinates and  $\gamma=1$ , using cylindrical coordinates at an axial-symmetric motion ( $\mathbf{x}=\mathbf{r}$ ,  $\mathbf{y}=\mathbf{z}$ ). By integrating the obtained expression by parts, we obtain the following matrix equation for definition of a finite element:

$$[k^{(e)}]\{\psi\} = \{f^{(e)}\}, \quad (11.42)$$

$$\text{where } [k^{(e)}] = \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \frac{2\gamma}{x} [N]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) x^\gamma dA; \quad (11.43)$$

$$\{f^{(e)}\} = \int_{\Gamma^{(e)}} [N]^T x^\gamma \left( \frac{\partial \psi}{\partial x} l_x + \frac{\partial \psi}{\partial y} l_y \right) d\Gamma; \quad l_x, l_y - \text{direction cosines} \quad (11.44)$$

For the solution of the given problem the following *boundary conditions* are used: (1) the Dirichlet's condition – on a contour  $\Gamma_\psi$  a value of a stream function of a fluid is known, namely:

$$\psi = \psi_0 \quad (11.45)$$

and (2) the Neumann's condition – derivative of stream function of a fluid along a normal line to a contour  $\Gamma_n$  (fig. 11.6) is known, namely:

$$\frac{\partial \psi}{\partial n} = \frac{\partial \psi}{\partial x} |_x + \frac{\partial \psi}{\partial y} |_y \quad (11.46)$$

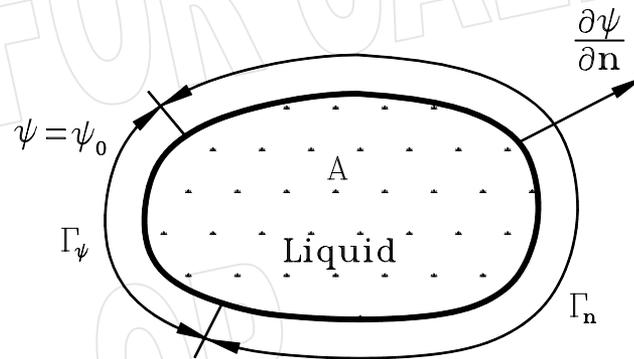


Fig. 11.6. Boundary conditions for the solution of the Laplace's equation, written in variables of stream function

Derivatives of functions of the forms over the global coordinates  $X$  and  $Y$ , incoming into a matrix  $[k^{(e)}]$ , and also elementary area and length of a contour are defined under the formulas (11.24) - (11.29). By taking into account relations (11.42), the global equations system of the following view is formed:

$$[K]\{\Phi\} = \{F\} \quad (11.47)$$

It is necessary to note, that in case of a motion of a fluid in Cartesian axials, matrix  $[K]$  is *symmetric*, while in case of a motion in cylindrical axials – *asymmetrical*.

### 11.2.2. Input data for the solution of the problem

For the solution of the Laplace's equation describing a *nonvortical motion of an incompressible fluid*, the *Stream\_flow* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order.

In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions (*nbond*) and the number of Neumann's boundary conditions (*ngrad*). In the *second* line, the following parameters are coded: number of nodes, in which the values of a stream function, and velocity of a motion (*nlp*) are defined; the number of sides of finite elements via which a stream of a fluid passes on (*ndeb*); a type of a problem (*ngama*), and a print code of intermediate results (*kprint*). If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0; otherwise – in *cylindrical* frames an axial-symmetric problem is solved.

In the *third* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). if parameter *ksolve*=0, then for the solution, the *'solve'* function of the *Maple-*

language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode* =4} are coded. After of array **Mtop** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into the **Lbond** array the numbers of nodes, in which are known Dirichlet's boundary conditions, are coded.

Behind of array **Lbond** into the data file the elements of **Lgrad**(*ngrad*, 3) array are recorded line by line. Into each line of the file the line number and elements of **Lgrad** array are recorded. Into the *first* column of the **Lgrad** array the number of a finite element, while into the *second* and the *third* column the numbers of nodes of the element side on which the Neumann's boundary conditions are known, are recorded.

Behind of array **Lgrad** into the data file the numbers of nodes, in which are defined the values of a stream function and velocity (array **Lpoint**(*nlp*)) are recorded line by line. Into each line of the file the line number and elements of **Lpoint** array and number of a node are recorded. Behind of array **Lpoint** into the data file the elements of **Ldebitas**(*ndeb*, 2) array are recorded line by line. Into this array the numbers of nodes of the sides of finite elements are coded, via which a stream of a fluid passes. Into each line, a serial number of line **Ldebitas** and two nodes of a side are recorded.

Behind of array **Ldebitas** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, into the *second* column of the **Coord** array the *y*-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded.

Behind of array **Coord** the elements of **Bond**(*nbond*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions, i.e. the values of a *stream function*, are recorded. Into each line of the datafile a line number of **Bond** array and a boundary value of a *stream function* are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

In each subsequent *ngrad* lines of the datafile, the elements of the **Grad**(*ngrad*, 4) array are recorded line by line. Into the *first* two columns of this array the values of partial derivatives

$\left( \frac{\partial \psi}{\partial x} \text{ and } \frac{\partial \psi}{\partial x} \right)_i$  in *i*-th node of a side, and into the *third* and the *fourth* – the value of partial

derivatives  $\left( \frac{\partial \psi}{\partial x} \text{ and } \frac{\partial \psi}{\partial x} \right)_j$  in *j*-th node of a side of a finite element, are recorded. In the datafile

between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbond, ngrad, nlp, ndeb, ngama, kprint, ksolve, niter, toler*  
 Text line \*  
 Arrays **Mtop**(*nelem*, *nnode*)  
 Text line \*  
 Array **Lbond**(*nbond*)  
 Text line \*  
 Array **Lgrad**(*ngrad*, 3)  
 Text line \*

Array *Lpoint*(*nlp*)  
 Text line \*  
 Array *Ldebitas*(*ndeb*, 2)  
 Text line \*  
 Array *Coord*(*npoin*, *ndime*)  
 Text line \*  
 Array *Bond*(*nbond*)  
 Text line \*  
 Array *Grad*(*ngrad*, 4)

### 11.2.3. Brief description of the Stream\_flow program solving the problem

The *Stream\_flow* program was programmed on the *Maple*-language; it consists of the basic program and 31 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of *potential of velocity*, *projection of velocities* of a motion of a fluid onto axes **X** and **Y**, and also a *hydrodynamic pressure* in nodes of finite elements. The calculation results are output on the monitor and are recorded into a file, for that to variables *file\_rez1* and *file\_rez2* of the program must be ascribed qualifiers of the target files. Into the file *file\_rez1* values of a *stream function* in nodes of finite elements and a *consumption of a fluid*, while into the file *file\_rez2* – values of a *stream function* and *velocities* in nodes, indicated in the array *Lpoint*, are recorded.

### 11.2.4. An example of use of the Maple-program Stream\_flow

The problem of flowing around of a cylinder by a flat stream between parallel walls is considered. Boundaries of explored doubly connected area of a fluid are a surface of the cylinder, the limitative walls and conventionally chosen cross-sections located symmetrically concerning an axis of the cylinder. The stream has two symmetry axes only, therefore it is enough to consider the area **ABCDE**, presented on the *fig.* 11.7.

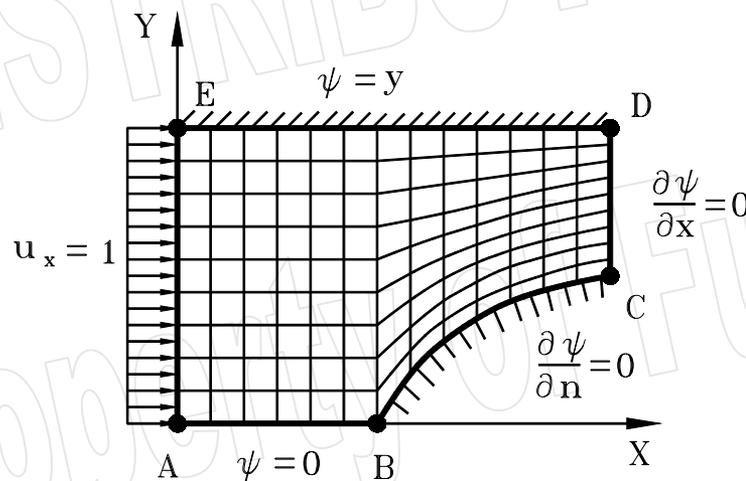


Fig. 11.7. Flowing around of the cylinder by a stream between parallel walls

On the symmetry axis **AB** the stream function is equal  $\psi = 0$ , on boundaries **BC** and **CD** the Neumann's conditions  $\frac{\partial \psi}{\partial n} = 0$  and  $\frac{\partial \psi}{\partial x} = 0$  are known, accordingly. On the boundary **AE** the

stationary velocity of an input stream of a fluid  $u_x = 1 \text{ m/s}$  is known. From expressions (11.33) the values of a stream function on boundaries EA and ED are determined, i.e. takes place the

following relation:  $\int_0^y d\psi = \int_0^y u_x dy$ , which after suitable transformations accepts the following form:

$$\psi = u_x y = y \tag{11.48}$$

*Input data for the test example:* a volume of a fluid between the parallel walls is divided onto  $n_{elem}=117$  finite elements, number of nodes  $n_{poin}=140$ , number of boundary conditions of the Dirichlet  $n_{bond}=36$ , number of boundary conditions of the Neumann  $n_{grad}=1$ , radius of a cylinder 1 m; distance from axis of the cylinder up to a wall 2 m; length of a pipe 3.5 m,  $k_{print}=0$ ,  $ksolve=1$ ,  $ngama=0$ ,  $nlp=10$ ,  $ndeb=9$ ,  $toler=10^{(-6)}$ ,  $niter=50$ ; the stream of a fluid via the boundary CD is supposed by equal  $Q=2.0 \text{ m}^3/\text{sec}$ .

**Results of calculation over the test example:**

The calculated values of a stream function (value) in nodes (node) of the finite elements accept the following form:

node=1	value=0e-01	node=28	value=2.489221e-01
node=2	value=0e-01	node=29	value=8.888e-01
node=3	value=0e-01	node=30	value=8.702682e-01
node=4	value=0e-01	node=31	value=8.237279e-01
node=5	value=0e-01	node=32	value=7.334017e-01
node=6	value=0e-01	node=33	value=6.145532e-01
node=7	value=0e-01	node=34	value=5.262543e-01
node=8	value=2.222e-01	node=35	value=4.726658e-01
node=9	value=2.157223e-01	node=36	value=1.1111e+00
node=10	value=1.988524e-01	node=37	value=1.092494e+00
node=11	value=1.622823e-01	node=38	value=1.046916e+00
node=12	value=1.035321e-01	node=39	value=9.625620e-01
node=13	value=4.851642e-02	node=40	value=8.589501e-01
node=14	value=8.182105e-03	node=41	value=7.866147e-01
node=15	value=4.444e-01	node=42	value=7.444784e-01
node=16	value=4.322391e-01	node=43	value=1.3333e+00
node=17	value=4.008403e-01	node=44	value=1.316963e+00
node=18	value=3.344361e-01	node=45	value=1.277667e+00
node=19	value=2.336068e-01	node=46	value=1.208032e+00
node=20	value=1.475905e-01	node=47	value=1.127576e+00
node=21	value=9.102237e-02	node=48	value=1.074540e+00
node=22	value=6.666e-01	node=49	value=1.044840e+00
node=23	value=6.502597e-01	node=50	value=1.5555e+00
node=24	value=6.085729e-01	node=51	value=1.543508e+00
node=25	value=5.237684e-01	node=52	value=1.514799e+00
node=26	value=4.038315e-01	node=53	value=1.465694e+00
node=27	value=3.088556e-01	node=54	value=1.411630e+00

node=55	value=1.377563e+00	node=98	value=1.772119e+00
node=56	value=1.359048e+00	node=99	value=2e+00
node=57	value=1.7777e+00	node=100	value=2e+00
node=58	value=1.771378e+00	node=101	value=0e-01
node=59	value=1.756259e+00	node=102	value=0e-01
node=60	value=1.730983e+00	node=103	value=2.257461e-01
node=61	value=1.703994e+00	node=104	value=2.527354e-01
node=62	value=1.687448e+00	node=105	value=4.555680e-01
node=63	value=1.678599e+00	node=106	value=4.943314e-01
node=64	value=2e+00	node=107	value=6.843862e-01
node=65	value=2e+00	node=108	value=7.259505e-01
node=66	value=2e+00	node=109	value=9.105170e-01
node=67	value=2e+00	node=110	value=9.496793e-01
node=68	value=2e+00	node=111	value=1.133147e+00
node=69	value=2e+00	node=112	value=1.166972e+00
node=70	value=2e+00	node=113	value=1.352870e+00
node=71	value=0e-01	node=114	value=1.379357e+00
node=72	value=9.659793e-02	node=115	value=1.569951e+00
node=73	value=2.512640e-01	node=116	value=1.588025e+00
node=74	value=4.531172e-01	node=117	value=1.785491e+00
node=75	value=6.874426e-01	node=118	value=1.794576e+00
node=76	value=9.405394e-01	node=119	value=2e+00
node=77	value=1.202190e+00	node=120	value=2e+00
node=78	value=1.467213e+00	node=121	value=0e-01
node=79	value=1.733413e+00	node=122	value=0e-01
node=80	value=2e+00	node=123	value=2.685413e-01
node=81	value=0e-01	node=124	value=2.737782e-01
node=82	value=0e-01	node=125	value=5.167287e-01
node=83	value=1.433613e-01	node=126	value=5.242466e-01
node=84	value=1.881221e-01	node=127	value=7.502245e-01
node=85	value=3.303062e-01	node=128	value=7.581311e-01
node=86	value=4.001532e-01	node=129	value=9.724429e-01
node=87	value=5.463044e-01	node=130	value=9.799707e-01
node=88	value=6.244143e-01	node=131	value=1.186644e+00
node=89	value=7.790584e-01	node=132	value=1.193210e+00
node=90	value=8.536513e-01	node=133	value=1.394676e+00
node=91	value=1.020055e+00	node=134	value=1.399920e+00
node=92	value=1.084386e+00	node=135	value=1.598708e+00
node=93	value=1.264355e+00	node=136	value=1.602242e+00
node=94	value=1.314544e+00	node=137	value=1.799909e+00
node=95	value=1.509576e+00	node=138	value=1.801791e+00
node=96	value=1.543872e+00	node=139	value=2e+00
node=97	value=1.754843e+00	node=140	value=2e+00

The values of a stream function (*stream*) and a velocity  $u_x$  (*Velx*) in nodes (*node*) of finite elements in a direction of axis **X** along boundary **CD** are represented as follows:

node=122	y=1e+00	stream=0e-01	velx=2.464250e+00
node=124	y=1.1111e+00	stream=2.737782e-01	velx=2.254441e+00
node=126	y=1.2222e+00	stream=5.242466e-01	velx=2.105170e+00
node=128	y=1.3333e+00	stream=7.581311e-01	velx=1.996756e+00
node=130	y=1.4444e+00	stream=9.799707e-01	velx=1.917620e+00
node=132	y=1.5556e+00	stream=1.193210e+00	velx=1.860582e+00
node=134	y=1.6667e+00	stream=1.399920e+00	velx=1.821073e+00
node=136	y=1.7778e+00	stream=1.602242e+00	velx=1.796125e+00
node=138	y=1.8889e+00	stream=1.801791e+00	velx=1.784053e+00

The allocation of velocity  $u_x$  in the direction of axis **X** along the boundary **CD** is given on the fig. 11.8.

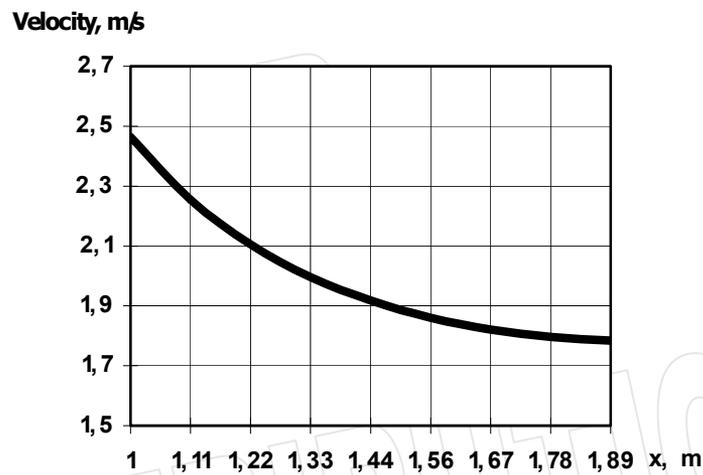


Fig. 11.8. The allocations of velocity  $u_x$  along the boundary **CD**

While the allocation of a stream function  $\psi$  in volume of a fluid is represented by isolines on the fig. 11.9.

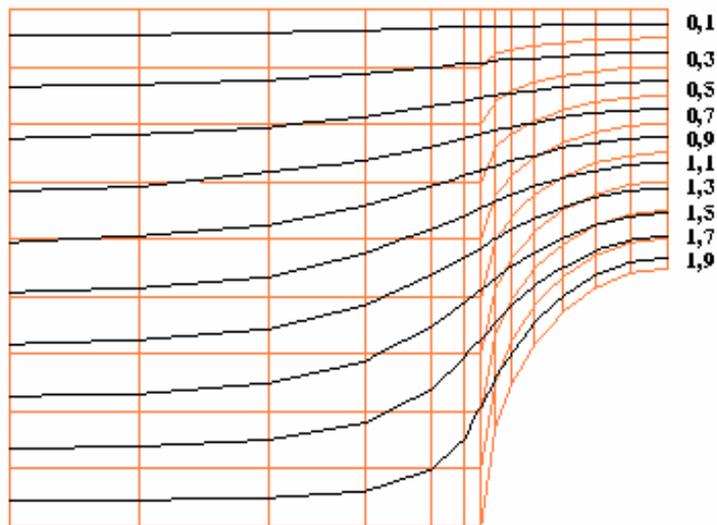


Fig. 11.9. Isolines of a stream function in volume of a fluid

The *Stream\_flow* program is intended for the solution of two-dimensional Laplace's equation written for *variable stream function* in cartesian or cylindrical axials for a case of an asymmetrical problem. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles *Mb4\_2\_new.mws*, *Mb4\_2.dat* and *Mb4\_2\_1.rez* accordingly.

### 11.3. The Navier-Stokes's equations represented in the terms of velocity-pressure

The common motion of a fluid is described by the equations of Navier-Stokes. These equations are used for describing of a motion of *viscous compressible* fluids and gases. From the considered equations, as the partial case we can obtain the equations of motion for *non-viscous* and *incompressible* fluids and gases.

#### 11.3.1. The calculated equations of Navier-Stokes

For a homogeneous incompressible viscous fluid the equations system of Navier-Stokes accepts the following form [50]:

$$\frac{\partial \bar{\mathbf{U}}}{\partial t} + (\bar{\mathbf{U}} \nabla) \bar{\mathbf{U}} = -\frac{1}{\rho} \text{grad } p + \nu \Delta \bar{\mathbf{U}} + \mathbf{f} \bar{\mathbf{n}}; \quad \text{div } \bar{\mathbf{U}} = 0 \quad (11.49)$$

In the equations system (11.49) by the sought functions appear a vector of velocity  $\bar{\mathbf{U}}$  and pressure  $p$ , which depend on space coordinates and time  $t$ . Parameters are density  $\rho$ , kinematic viscosity coefficient  $\nu = \frac{\mu}{\rho}$  ( $\mu$ - coefficient of dynamic viscosity),  $\mathbf{f}$  - strength function, and  $\bar{\mathbf{n}}$  - unit vector. The first of the equations in system (11.49), representing system of three equations for projections of a vector of velocity  $\bar{\mathbf{U}}(u_x, u_y, u_z)$ , is called as the *equation of a momentum* and represents a balance between *inertial forces, pressure, friction* and *forces of masses*. While the *second* from the equations in the system (11.49) is called as a *continuity equation* or an *incompressibility equation*. Two-dimensional motion of a fluid further is considered. Depending on a concrete physical situation the following kinds of boundary conditions are distinguished:

1. *Boundary conditions* on an impenetrable solid surface  $S$ , which are usually called as *adhesion conditions*. If velocity of a motion of a surface is  $\bar{\mathbf{U}}_c$ , the boundary condition on this surface accepts the following view:  $\bar{\mathbf{U}}|_S = \bar{\mathbf{U}}_c$  (11.50)

2. *Boundary condition* far off from a streamline body, which has an asymptotic character, namely:  $\bar{\mathbf{U}} \rightarrow \bar{\mathbf{U}}_\infty$  at  $\bar{r} \rightarrow \infty$ ; where  $\bar{r}$  - distance from a surface of a streamline body (11.51)

3. The *periodic boundary conditions* representing the especial type of boundary conditions, which are usually considered at streamline of an infinite sequence of iterant bodies. The parameters of a stream before a body are equal to parameters of stream behind the body, namely:  $\bar{\mathbf{U}}_1 = \bar{\mathbf{U}}_2$  (11.52)

4. The *symmetry conditions* representing an especial type of boundary conditions, arising owing to the certain suppositions about properties of symmetry of a current. At flowing around of a symmetric profile by an uniform stream under a zero angle of attack by natural boundary conditions are assumed the conditions on a symmetry axis, namely:  $u_y = 0$ ;  $\frac{\partial u_x}{\partial n} = 0$ , (11.53),

where:  $\mathbf{u}_y$  – constituent of velocity over a normal straight to a symmetry axis and  $\bar{\mathbf{n}}$  – direction, normal to a symmetry axis

5. The boundary conditions on an interface of two mediums (when the surface is fixed, and by friction in one of mediums can be neglected) of the following view:  $\mathbf{u}_n = \mathbf{0}$ ,  $\frac{\partial \mathbf{u}}{\partial n} = \mathbf{0}$  (11.54).

That is similar to conditions of symmetry (11.53). The conditions (11.54), against conditions of an adhesion (11.50), admit a motion of a fluid along a surface.

The equations (11.49) together with the initial conditions  $\mathbf{u}_{x_0} = \mathbf{u}_x(\mathbf{x}, \mathbf{y}, \mathbf{t} = \mathbf{0})$ ,  $\mathbf{u}_{y_0} = \mathbf{u}_y(\mathbf{x}, \mathbf{y}, \mathbf{t} = \mathbf{0})$ ,  $\mathbf{p}_0 = \mathbf{p}(\mathbf{x}, \mathbf{y}, \mathbf{t} = \mathbf{0})$  and corresponding boundary conditions represent the closed system allowing to define the fields of velocity and pressure of a homogeneous incompressible viscous fluid and their change in a time. At realization of evaluations the dimensionless form of record of input equations, initial and boundary conditions is usually used. By choosing as a scale of velocity and length  $\mathbf{V}$  and  $\mathbf{L}$  accordingly, and for parameters  $\rho$ ,  $\mathbf{v}$ ,  $\mathbf{f}$  – their values, given by the task conditions, it is possible to write the initial equations system (11.49) in the dimensionless form as follows:

$$\frac{\partial \bar{\mathbf{U}}}{\partial \bar{\mathbf{t}}} + (\bar{\mathbf{U}} \nabla) \bar{\mathbf{U}} = -\text{grad} \bar{\mathbf{p}} + \frac{1}{\text{Re}} \Delta \bar{\mathbf{U}} + \mathbf{F} \cdot \bar{\mathbf{n}} \quad (11.55)$$

$$\text{div} \bar{\mathbf{U}} = \mathbf{0} \quad (11.56)$$

where:  $\text{Re} = \frac{\mathbf{VL}}{\gamma}$  – Reynolds number representing the ratio of inertial forces to forces of viscosity and defining an intensity of a forced convection;  $\mathbf{F} = \mathbf{F}_0 \mathbf{f}$ ,  $\mathbf{F}_0 = \frac{1}{\sqrt{\text{Fr}}}$  – likeness criterion representing the ratio of forces of masses to inertial forces;  $\mathbf{F}_0 = \frac{\mathbf{fL}}{\mathbf{v}^2}$ ,  $\text{Fr}$  – the Froude number. For parameters of time  $\bar{\mathbf{t}}$  and pressure  $\bar{\mathbf{p}}$  in the system (11.55) - (11.56) the scales  $\frac{\mathbf{L}}{\mathbf{v}}$  and  $\rho \mathbf{V}^2$  are used accordingly.

Use of dimensionless system pursues two purposes: reduction of values of the calculated quantities to the corresponding scale, and also calculation and handling of the results in the common criterial form. Further in the text the dimensionless quantities are designated without their underlining. Then, the equation of Navier-Stokes in Bussinesk's approximation, written in the dimensionless form, accepts the following form:

$$\frac{\partial u_x}{\partial t} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{\text{Re}} \left( \Delta u_x - \frac{\gamma}{x^2} u_x \right) = 0; \quad \frac{\partial u_y}{\partial t} + u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{\text{Re}} \Delta u_y = 0; \quad (11.57)$$

$$\frac{\partial u_x}{\partial x} + \frac{\gamma}{x} u_x + \frac{\partial u_y}{\partial y} = 0$$

where  $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\gamma}{x} \frac{\partial}{\partial x} + \frac{\partial^2}{\partial y^2} = \mathbf{0}$  – Laplacian of scalar function;  $\gamma = \mathbf{0}$ , using cartesian coordinates, and  $\gamma = \mathbf{1}$ , using cylindrical coordinates ( $\mathbf{x} = \mathbf{r}$ ,  $\mathbf{y} = \mathbf{z}$ ) in case of an axial-symmetric problem. The equations system (11.57) for stationary process is solved by means of the FEM. The velocities in limits of a finite element are approximated by the next expressions:

$$u_x = \sum_{i=1}^n N_i(\mathbf{r}, \mathbf{s}) u_i = [\mathbf{N}]\{\mathbf{u}\}; \quad u_y = \sum_{i=1}^n N_i(\mathbf{r}, \mathbf{s}) v_i = [\mathbf{N}]\{\mathbf{v}\}$$

while the *pressure* is approximated by expression of the following form:

$$\mathbf{p} = \sum_{i=1}^{n_p} \mathbf{N}_{pi}(r, s) p_i = [\mathbf{N}_p] \{\mathbf{p}\} \quad (11.58)$$

where  $\mathbf{N}_i$ ,  $\mathbf{N}_{pi}$  – shape functions for velocities and pressures;  $\{\mathbf{u}\}$ ,  $\{\mathbf{v}\}$ ,  $\{\mathbf{p}\}$  – vectors of nodal velocities and pressures. Application of the Galerkin's method and orthogonalization of discrepancies to shapes functions  $\mathbf{N}_i$  and  $\mathbf{N}_{pi}$  reduce to the following equations system:

$$\int_{A^{(e)}} [\mathbf{N}]^T \left[ u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{\text{Re}} \left( \Delta u_x - \frac{\gamma}{x^2} u_x \right) \right] dA = 0;$$

$$\int_{A^{(e)}} [\mathbf{N}]^T \left[ u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{\text{Re}} \Delta u_y \right] dA = 0; \quad \int_{A^{(e)}} [\mathbf{N}_p]^T \left( \frac{\partial u_x}{\partial x} + \frac{\gamma}{x} u_x + \frac{\partial u_y}{\partial y} \right) dA = 0. \quad (11.59)$$

By applying the Green's formula and by substituting expressions (11.58) into (11.59), we obtain the following equations system:

$$[\mathbf{k}^{(e)}] \{\mathbf{r}\} = \{\mathbf{f}^{(e)}\}, \quad (11.60)$$

$$\text{where } [\mathbf{k}^{(e)}] = \begin{bmatrix} k_{uu} & k_{up} & 0 \\ k_{pu} & 0 & k_{pv} \\ 0 & k_{vp} & k_{vv} \end{bmatrix}; \quad \{\mathbf{f}^{(e)}\} = \begin{Bmatrix} f_u \\ 0 \\ f_v \end{Bmatrix};$$

$$[k_{uu}] = \int_{A^{(e)}} \left[ [\mathbf{N}]^T \left( u_x \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + u_y \left[ \frac{\partial \mathbf{N}}{\partial y} \right] \right) + \frac{1}{\text{Re}} \left( \left[ \frac{\partial \mathbf{N}}{\partial x} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + \left[ \frac{\partial \mathbf{N}}{\partial y} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial y} \right] + \frac{1}{x^{2\gamma}} [\mathbf{N}]^T [\mathbf{N}] \right) \right] dA;$$

$$[k_{up}] = \int_{A^{(e)}} [\mathbf{N}]^T \left[ \frac{\partial \mathbf{N}_p}{\partial x} \right] dA; \quad [k_{pu}] = \int_{A^{(e)}} [\mathbf{N}_p]^T \left( \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + \frac{\gamma}{x} [\mathbf{N}] \right) dA; \quad [k_{pv}] = \int_{A^{(e)}} [\mathbf{N}_p]^T \left[ \frac{\partial \mathbf{N}}{\partial y} \right] dA;$$

$$[k_{vp}] = \int_{A^{(e)}} [\mathbf{N}]^T \left[ \frac{\partial \mathbf{N}_p}{\partial y} \right] dA; \quad (11.61)$$

$$[k_{vv}] = \int_{A^{(e)}} \left[ [\mathbf{N}]^T \left( u_x \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + u_y \left[ \frac{\partial \mathbf{N}}{\partial y} \right] \right) + \frac{1}{\text{Re}} \left( \left[ \frac{\partial \mathbf{N}}{\partial x} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + \left[ \frac{\partial \mathbf{N}}{\partial y} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial y} \right] \right) \right] dA;$$

$$\{f_u\} = \frac{1}{\text{Re}} \int_{\Gamma^{(e)}} [\mathbf{N}]^T x^\gamma \left( \frac{\partial u_x}{\partial x} l_x + \frac{\partial u_x}{\partial y} l_y \right) d\Gamma; \quad \{f_v\} = \frac{1}{\text{Re}} \int_{\Gamma^{(e)}} [\mathbf{N}]^T x^\gamma \left( \frac{\partial u_y}{\partial x} l_x + \frac{\partial u_y}{\partial y} l_y \right) d\Gamma;$$

$l_x$ ,  $l_y$  – direction cosines. For a choice of an approximation type of velocities and pressure we shall consider a *stress tensor* acting in a viscous fluid:

$$\sigma_{ij} = -p \delta_{ij} + \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \quad (11.62)$$

The components of this tensor depend on derivatives of velocity and pressure. For representation of velocity it is necessary to use shape functions of the higher order, than for pressure. If  $\mathbf{N}_{pi}$  – the linear functions, then  $\mathbf{N}_i$  should be quadratic. If that not is taken into account, then for a number of problems the incompatible equations system can be obtained. Therefore, for approximation of pressure the quadrangular linear isoparametric finite element, and for approximation of velocities – quadrangular quadratic isoparametric finite element (fig. 11.10) are used accordingly.

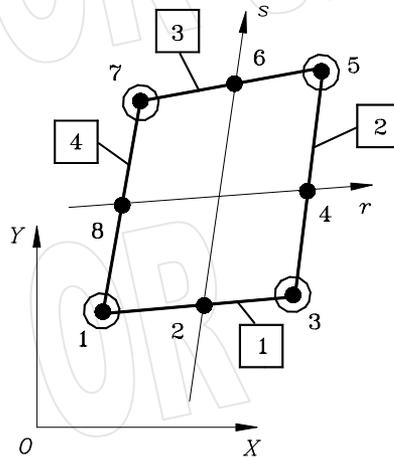


Fig. 11.10. A quadratic isoparametric finite element (by circles are marked nodes, in which the pressures are calculated, and in quadrates the numbers of the sides of finite elements are indicated)

With the help of the calculated relations (11.60) and (11.61) the global equations system of the following view are formed:

$$[\mathbf{K}]\{\Phi\} = \{\mathbf{F}\} \quad (11.63)$$

For increase of a stability of the solution on each iteration the solution is averaged, then the vector of unknowns is made more accurate, namely:

$$\{\mathbf{R}_{new}\} = (1 - \alpha)\{\mathbf{R}_{old}\} + \alpha\{\mathbf{R}_i\} \quad (11.64)$$

where:  $\alpha$  – relaxation coefficient ( $0 \leq \alpha \leq 1$ ).

### 11.3.2. Input data for solution of the problem

For the solution of the Navier-Stokes's equations describing a *stationary motion of an incompressible viscous fluid*, the *Navier\_Stokes* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoin*), the number of Dirichlet's boundary conditions (*nbond*) and the number of Neumann's boundary conditions (*ngrad*). In the *second* line, a print code of intermediate results (*kprint*), a type of a problem (*ngama*), the Reynolds number (*reinold*), dimensionless forces acting in direction of axes **X** and **Y** (*forcex*, *forcey*) are coded. If *kprint* = 0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0; otherwise – in *cylindrical* frames an axial-symmetric problem is solved.

In the *third* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). if parameter *ksolve*=0, then for the solution, the ``solve`` function of the Maple-language is used; otherwise, the equations system by the method of *conjugate gradients* (for case of *asymmetrical system*) is being solved. In the *fourth* line the number of iterations (*niteration*), tolerance (*tol*) and also a relaxation coefficient (*relax*) are recorded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*,*nnode*), where *nnode* - number of nodes of the finite element, i.e. *nnode* =8} are coded. After of array **Mtop** the elements of **Lbondv**(*nbondv*, 2) array are coded line by line. Into each line of the file the line number and this array, the number of a node and a degree of freedom in the node, in which the Dirichlet's boundary conditions are known, are recorded.

Behind of array **Lbondv** into the data file the elements of **Lgrad**(*ngrad*, 4) array are recorded line by line. Into each line of the file the line number of this array and three numbers of nodes, laying on one side of a finite element, are recorded. The numbers of the sides of a finite element are represented on the *fig. 11.10*. Behind of array **Lgrad** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* - dimensionality of the problem (*ndime*=2), are recorded line by line. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded. Behind of array **Coord** the elements of **Bondv**(*nbondv*) array are recorded line by line. Into each line of the datafile a line number of the **Bondv** array and a boundary value of corresponding variable are recorded.

In each subsequent *ngrad* lines of the datafile, the elements of the **Grad**(*ngrad*, 4) array are recorded line by line. Into each line of the datafile a line number of **Grad** array and values of partial derivatives  $\frac{\partial u_x}{\partial x}$ ,  $\frac{\partial u_y}{\partial x}$ ,  $\frac{\partial u_x}{\partial y}$  and  $\frac{\partial u_y}{\partial y}$  are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbondv, ngrad, kprint, ngama, ksolve, niter, toler, niteration, tol, relax*  
 Text line \*  
 Arrays **Mtop**(*nelem*, *nnode*)  
 Text line \*  
 Array **Lbondv**(*nbondv*, 2)  
 Text line \*  
 Array **Lgrad**(*ngrad*, 4)  
 Text line \*  
 Array **Coord**(*npoin*, *ndime*)  
 Text line \*  
 Array **Bondv**(*nbondv*)  
 Text line \*  
 Array **Grad**(*ngrad*, 4)

### 11.3.3. Brief description of the Navier\_Stokes program solving the problem

The *Navier\_Stokes* program was programmed on the Maple-language; it consists of the basic program and 38 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete

problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates *projections of velocities* of a motion of a fluid over axes **X, Y** and also a *pressure* in nodes of finite elements. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of projections of velocities and pressure in nodes of finite elements are recorded. To variables *file\_geo* and *file\_dr* of the program also must be ascribed the qualifiers of files, in which the *F\_isograf* procedure saves an information for visualization of the image of isolines of velocities.

### 11.3.4. An example of use of the Maple-program Navier\_Stokes

The *current of a fluid* in a closed square area, caused by a motion of one from its boundaries, is considered; other boundaries of the area remain fixed (fig. 11.11).

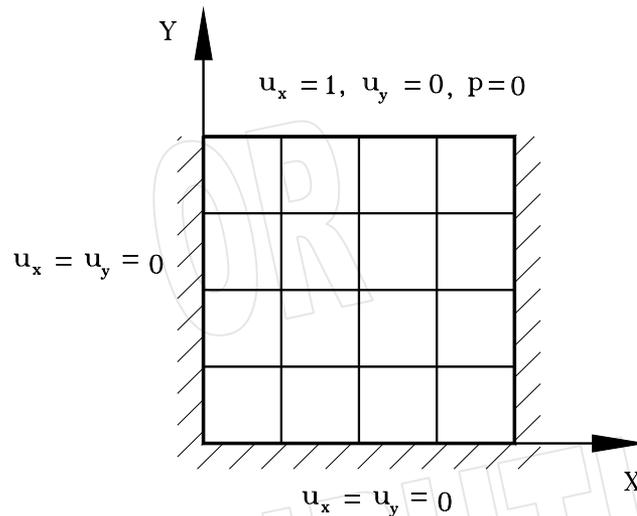


Fig. 11.11. The closed square area with a fluid and boundary conditions

The *boundary conditions* for the solution of the Navier-Stokes's equations accept the following form:

$u_x = 1, u_y = 0, p = 0$  - on a moving boundary;

$u_x = 0, u_y = 0$  - on the fixed boundaries.

Input data for the test example: a volume of a fluid is divided onto *nelem*=16 finite elements, number of nodes *npoin*=65, number of boundary conditions of the Dirichlet *nbondv*=69, number of boundary conditions of the Neumann *ngrad*=4, *kprint*=0, *ksolve*=1, *ngama*=0, *tol*=10<sup>-6</sup>, *niter*=20; *forcex*=0, *forcey*=0, *toler*=10<sup>-6</sup>, *niteration*=5, *relax*=0.5, Reynolds number *Re*=20, the sizes of area 0.5x0.5.

#### Results of calculation over the test example:

The values of projections of velocities over axes **X, Y** (**V<sub>x</sub>, V<sub>y</sub>**) and a pressure (**pressure**) in nodes (**node**) of finite elements accept the following foem:

node=1	velx=0	vely=0	pressure=-.043820
node=2	velx=0	vely=0	
node=3	velx=0	vely=0	pressure=-.090027
node=4	velx=0	vely=0	
node=5	velx=0	vely=0	pressure=.003254
node=6	velx=0	vely=0	

node=7	velx=0	vely=0	pressure=.095705
node=8	velx=0	vely=0	
node=9	velx=0	vely=0	pressure=.045046
node=10	velx=0	vely=0	
node=11	velx=-.014580	vely=.009599	
node=12	velx=-.054424	vely=.002835	
node=13	velx=-.014781	vely=-.006803	
node=14	velx=0	vely=0	
node=15	velx=0	vely=0	pressure=.139207
node=16	velx=.007008	vely=.022738	
node=17	velx=-.035111	vely=.038687	pressure=-.097741
node=18	velx=-.074634	vely=.023289	
node=19	velx=-.084452	vely=.004212	pressure=-.000590
node=20	velx=-.077237	vely=-.017938	
node=21	velx=-.038310	vely=-.033941	pressure=.106987
node=22	velx=.006662	vely=-.021019	
node=23	velx=0	vely=0	pressure=-.140962
node=24	velx=0	vely=0	
node=25	velx=-.059710	vely=.060479	
node=26	velx=-.104895	vely=.001810	
node=27	velx=-.062103	vely=-.058678	
node=28	velx=0	vely=0	
node=29	velx=0	vely=0	pressure=-.230199
node=30	velx=-.052759	vely=.131605	
node=31	velx=-.086635	vely=.092360	pressure=-.131688
node=32	velx=-.092210	vely=.055022	
node=33	velx=-.095602	vely=.000540	pressure=-.001696
node=34	velx=-.091402	vely=-.054577	
node=35	velx=-.086399	vely=-.092023	pressure=.131851
node=36	velx=-.054439	vely=-.131972	
node=37	velx=0	vely=0	pressure=.226984
node=38	velx=0	vely=0	
node=39	velx=-.066229	vely=.122204	
node=40	velx=-.027620	vely=-.001450	
node=41	velx=-.063450	vely=-.121919	
node=42	velx=0	vely=0	
node=43	velx=0	vely=0	pressure=-1.529297
node=44	velx=-.062275	vely=-.011489	
node=45	velx=.115278	vely=.005902	pressure=-.520806
node=46	velx=.152805	vely=-.001726	
node=47	velx=.142286	vely=-.003725	pressure=.005232
node=48	velx=.156959	vely=-.007290	
node=49	velx=.113314	vely=-.009200	pressure=.528644

node=50	velx=-.055837	vely=.015699	
node=51	velx=0	vely=0	pressure=1.559705
node=52	velx=0	vely=0	
node=53	velx=.410073	vely=-.107279	
node=54	velx=.544212	vely=-.009534	
node=55	velx=.408743	vely=.105881	
node=56	velx=0	vely=0	
node=57	velx=0	vely=0	pressure=0
node=58	velx=1	vely=0	
node=59	velx=1	vely=0	pressure=0
node=60	velx=1	vely=0	
node=61	velx=1	vely=0	pressure=0
node=62	velx=1	vely=0	
node=63	velx=1	vely=0	pressure=0
node=64	velx=1	vely=0	
node=65	velx=0	vely=0	pressure=0

The allocation of velocity  $u_x(x=0.25, y)$  along the vertical is represented on the fig. 11.12.

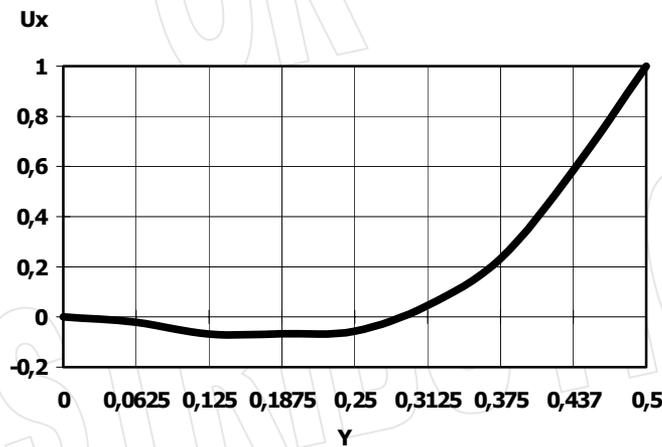


Fig. 11.12. The allocation of velocity  $u_x(x=0.25, y)$  along the vertical

While the allocation of velocity  $u_x$  in volume of fluid is represented by isolines on the fig. 11.13.

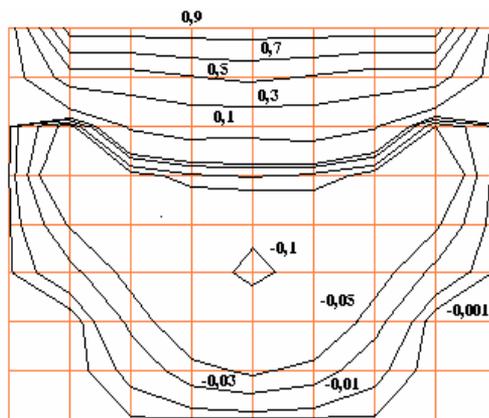


Fig. 11.13. Isolines of velocity  $u_x$  in volume of fluid

The *Navier\_Stokes* program is intended for the solution of two-dimensional Navier-Stokes's equations written for variables *velocity and pressure* in cartesian or cylindrical axials for case of an axial-symmetrical task. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of attached to the book in datafiles *Mb4\_3\_new.mws*, *Mb4\_3.dat* and *Mb4\_3\_1.rez* accordingly.

## 11.4. Solution of the Navier-Stokes's equations, written in the terms of variables of the vortex and stream function

The Navier-Stokes's equations, written in the terms of variables of the *vortex and stream function*, are widely used in applied problems of *hydromechanics*. That is conditioned by the circumstance, what the equations system of Navier-Stokes is divided into two equations: the non-stationary equation for a vortex and stationary Poisson equation for a stream function. These equations are solved separately, that in turn simplifies the solution of the *equations* of Navier-Stokes as a whole.

### 11.4.1. The calculated equations of Navier-Stokes in variables of vortex and stream function

The Navier-Stokes's equations in the Bussinesk's approximation, written in a variables of *vortex and stream function*, accept the following view [69]:

$$\frac{\partial \omega}{\partial t} + u_x \frac{\partial \omega}{\partial x} + u_y \frac{\partial \omega}{\partial y} - \frac{u_x}{x^\gamma} \omega - \frac{1}{\text{Re}} \left( \frac{\partial^2 \omega}{\partial x^2} + \frac{\gamma}{x} \frac{\partial \omega}{\partial x} + \frac{\partial^2 \omega}{\partial y^2} - \frac{\gamma}{x^2} \omega \right) = 0; \quad (11.65)$$

$$\frac{\partial^2 \psi}{\partial x^2} - \frac{\gamma}{x} \frac{\partial \psi}{\partial x} + \frac{\partial^2 \psi}{\partial z^2} = x^\gamma \omega \quad (11.66)$$

where:  $\gamma=0$ , using *cartesian* coordinates, and  $\gamma=1$ , using *cylindrical* axials for a case of an axial-symmetric motion ( $x=r, y=z$ ). *Stream function* and *vortex*  $\omega$  are given by the following relations:

$$u_x = \frac{1}{x^\gamma} \frac{\partial \psi}{\partial y}; \quad u_y = -\frac{1}{x^\gamma} \frac{\partial \psi}{\partial x}; \quad \omega = \frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x} \quad (11.67)$$

The link of a vortex with a stream function in view (11.66) can be received, substituting expression for a velocity written via a stream function, into the expression for a vortex (11.67). The equation for a vortex (11.65) is gained if to eliminate a pressure from equations of motion in variables *velocity - pressure* (11.49) and to use expression of a vortex in the form (11.67). The equations systems (11.65), (11.66) have one feature, namely: the dynamic boundary conditions of an adhesion on solid boundary accept the following form:

$$\psi = 0, \quad \frac{\partial \psi}{\partial n} = 0 \quad (11.68)$$

Thus, the boundary conditions for a *vortex* are not preset, what is one of features at the solution of the equations system. The equations system (11.65), (11.66) are solved by means of the FEM; in addition, the *quadrangular linear isoparametric* finite element is used (fig. 11.1). *Vortex and stream function* in a finite element are approximated by the following expressions:

$$\omega(r, s) = \sum_{i=1}^n N_i(r, s) \{\omega\} = [N] \{\omega\}; \quad (11.69)$$

$$\psi(\mathbf{r}, \mathbf{s}) = \sum_{i=1}^n \mathbf{N}_i(\mathbf{r}, \mathbf{s}) \{\psi\} = [\mathbf{N}] \{\psi\} \quad (11.70)$$

where:  $[\mathbf{N}(\mathbf{r}, \mathbf{s})] = [\mathbf{N}_1, \mathbf{N}_2, \dots, \mathbf{N}_n]$  - matrix of shape functions;  $\{\omega\}$ ,  $\{\psi\}$  - vectors of nodal values of a vortex and stream function, accordingly. By substituting expressions (11.69), (11.70) into the equations system (11.65) - (11.66) and by applying the Galerkin's procedure, we shall receive the following *matrix equations* for a finite element:

$$[\mathbf{k}_1^{(e)}] \{\psi\} = \{\mathbf{f}_1^{(e)}\}; \quad (11.71)$$

$$[\mathbf{m}^{(e)}] \{\dot{\omega}\} + [\mathbf{k}_2^{(e)}] \{\omega\} = \{\mathbf{f}_2^{(e)}\}, \quad \text{where } [\mathbf{m}^{(e)}] = \int_{A^{(e)}} [\mathbf{N}]^T [\mathbf{N}] \, dA; \quad (11.72)$$

$$[\mathbf{k}_1^{(e)}] = \int_{A^{(e)}} \left( \left[ \frac{\partial \mathbf{N}}{\partial x} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + \left[ \frac{\partial \mathbf{N}}{\partial y} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial y} \right] + 2 \frac{\gamma}{x} [\mathbf{N}]^T \left[ \frac{\partial \mathbf{N}}{\partial x} \right] \right) dA; \quad (11.73)$$

$$[\mathbf{k}_2^{(e)}] = [\mathbf{k}_{21}^{(e)}] + [\mathbf{k}_{22}^{(e)}]; \quad (11.74)$$

$$[\mathbf{k}_{21}^{(e)}] = \int_{A^{(e)}} [\mathbf{N}]^T \left( u_x \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + u_y \left[ \frac{\partial \mathbf{N}}{\partial y} \right] - \frac{1}{x^\gamma} [\mathbf{N}]^T u_x [\mathbf{N}] \right) dA; \quad (11.75)$$

$$[\mathbf{k}_{22}^{(e)}] = \frac{1}{\text{Re}} \int_{A^{(e)}} \left( \left[ \frac{\partial \mathbf{N}}{\partial x} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial x} \right] + \left[ \frac{\partial \mathbf{N}}{\partial y} \right]^T \left[ \frac{\partial \mathbf{N}}{\partial y} \right] - \frac{1}{x^{2\gamma}} [\mathbf{N}]^T [\mathbf{N}] \right) dA; \quad (11.76)$$

$$\{\mathbf{f}_1^{(e)}\} = \{\mathbf{f}_{11}^{(e)}\} + \{\mathbf{f}_{12}^{(e)}\}; \quad (11.77)$$

$$\{\mathbf{f}_{11}^{(e)}\} = \int_{\Gamma^{(e)}} [\mathbf{N}]^T x^\gamma \left( \frac{\partial \psi}{\partial x} l_x + \frac{\partial \psi}{\partial y} l_y \right) d\Gamma; \quad (11.78)$$

$$\{\mathbf{f}_{12}^{(e)}\} = \int_{A^{(e)}} [\mathbf{N}]^T x^\gamma \omega \, dA; \quad (11.79)$$

$$\{\mathbf{f}_2^{(e)}\} = \int_{\Gamma^{(e)}} [\mathbf{N}]^T x^\gamma \left( \frac{\partial \omega}{\partial x} l_x + \frac{\partial \omega}{\partial y} l_y \right) d\Gamma. \quad (11.80)$$

By taking into account relations (11.71) and (11.72), the *global equations system* of the following view are formed:

$$[\mathbf{K}_1] \{\Psi\} = \{\mathbf{F}_1\}; \quad (11.81)$$

$$[\mathbf{M}] \{\dot{\Omega}\} + [\mathbf{K}_2] \{\Omega\} = \{\mathbf{F}_2\} \quad (11.82)$$

The boundary conditions (11.68) should be supplemented with boundary conditions for a vortex. The value of a vortex on boundary is determined from the next expression:

$$\Omega_{\Gamma,0} = \frac{2\Psi_1}{h^2} \quad (11.83)$$

where  $\Psi_1$  – value of a stream function in a point, which is remote on distance  $h$  perpendicular to a boundary. The approximate boundary condition closes the equations system (11.81) - (11.82). Use of approximate boundary conditions reduces to decreasing of a stability of solution of the equations system, for increasing of which the relaxation method is used, according to which the values of a vortex on boundary are represented as follows [69]:

$$\Omega_{\Gamma, i+1} = \alpha \left( \frac{2\Psi_{1, i+1}}{h^2} \right) + (1-\alpha) \Omega_{\Gamma, i}, \quad (11.84)$$

where:  $\alpha$  – relaxation coefficient ( $0 \leq \alpha \leq 1$ ). For a *non-stationary conditions* the use of a *relaxation* reduces to an additional discrepancy, which is proportional to quantity  $\Omega_{\Gamma, i+1} - \Omega_{\Gamma, i}$ . For elimination of the given discrepancy is introduced an interior iterative cycle, in which on each time stratum together with the solution of the equations of a vortex and stream function the relaxation of boundary conditions for a vortex is carried out. The interior iterations are carried out up to realization of the following conditions:

$$|\Omega_{\Gamma, i+1} - \Omega_{\Gamma, i}| < \text{tol} \quad (11.85)$$

The non-stationary equation of a vortex is integrated in a time, using the method of trapezoids, namely:

$$\Omega_{t+\tau} - \Omega_t = \frac{\tau}{2} (\dot{\Omega}_t + \dot{\Omega}_{t+\tau}) \quad (11.86)$$

where:  $\tau$  – integration step. Then the equation (11.82) can be written as follows:

$$\left( \frac{2}{\tau} [\mathbf{M}] + [\mathbf{K}] \right) \{ \Omega_{t+\tau} \} = \{ \mathbf{F}_{2, t+\tau} \} + [\mathbf{M}] \left( \frac{2}{\tau} \{ \Omega_t \} + \{ \dot{\Omega}_t \} \right) \quad (11.87)$$

The process of integration of the Navier-Stokes's equations in a time consists in solution of the equations systems (11.81) and (11.87) on each step in a time. The process of integration of the above equations is being continued up to realization of the following condition:

$$\max |\dot{\Omega}_{t+\tau, i} - \dot{\Omega}_{t, i}| < \varepsilon \quad (11.88)$$

i.e., when the process of a motion of a fluid becomes *steady-state (stationary)*.

#### 11.4.2. Input data for the solution of the problem

For the solution of the Navier-Stokes's equations, which are written in variables vortex - stream function and describe a nonvortical motion of an incompressible viscous fluid, the *Vort\_stream* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable  $\mathbf{F}$ . The data in the file are placed in the strict order. In the *first* line the number of finite elements (*parameter nelen*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions on a free surface of a fluid for variable of stream function (*nbsr*), the number of Dirichlet's boundary conditions on a free surface of a fluid for variable of vortex (*nbsn*), the number of sides of finite elements on fixed boundary

(*nwall*), the number of Neumann's boundary conditions for stream function (*ngrad*), and the number of Neumann's boundary conditions for vortex function (*ngradv*) are coded.

In the *second* line of the datafile, the following parameters are coded: the number of coordinate (*kcoord*) and number of nodes, in which the values of variable are printed (*nlp*). The number of *x*-coordinate by parameter *kcoord*=1, and the number of *y*-coordinate by parameter *kcoord*=2 are coded accordingly. In the *third* line, the number of iterations of an interior cycle (*niteration*), a relaxation coefficient (*relax*) and tolerance (*tol*) are coded.

In the *fourth* line of the datafile, a print code of intermediate results (*kprint*), a type of the problem (*ngama*) and the Reynolds number (*reinold*) are recorded. If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0; otherwise - in *cylindrical* frames an axial-symmetric problem is solved.

In the *fifth* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). if parameter *ksolve*=0, then for the solution, the ``solve`` function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved. In the *sixth* line of the datafile, the number of integration steps in a time (*ntime*) and an integration step (*dtime*) are recorded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*,*nnode*), where *nnode* - number of nodes of the finite element, i.e. *nnode* =4} are coded. After of array **Mtop** the elements of **Lbdsr**(*nbsr*) array are coded line by line. Into each line of the file the line number and element of **Lbdsr** array are recorded. Into the **Lbdsr** array the numbers of nodes, in which are known Dirichlet's boundary conditions on a free surface of a fluid for variable of stream function, are coded. Into each line of the datafile the line number of the **Lbdsr** array and the node number, in which are known Dirichlet's boundary conditions, are recorded. If the Dirichlet's boundary conditions on a free surface of a fluid for variable of stream function are not preset, then *nbsr*=1 and **Lbdsr**[1] = -1.

After of array **Lbdsr** the elements of **Lbdsn**(*nbsn*) array are coded line by line. Into each line of the file the line number and element of **Lbdsn** array are recorded. Into the **Lbdsn** array the numbers of nodes, in which are known Dirichlet's boundary conditions on a free surface of a fluid for variable of vortex, are coded. Into each line of the datafile the line number of the **Lbdsn** array and the node number, in which are known Dirichlet's boundary conditions, are recorded. If the Dirichlet's boundary conditions on a free surface of a fluid for vortex are not preset, then *nbsn*=1 and **Lbdsn**[1] = -1.

After of array **Lbdsn** the elements of **Mwall**(*nwal*, 3) array are coded line by line. Into each line of the file, the line number of the **Mwall** array, and the number of a finite element and numbers of nodes of a side of the finite element, which are on the fixed surface, are recorded. Behind of array **Mwall** into the data file the elements of **Lgrad**(*ngrad*, 3) array are recorded line by line. Into each line of the file, the line number of the **Lgrad** array, number of a finite element and numbers of nodes of the element, in which the Neumann's boundary conditions for variable of stream function are given, are recorded.

Behind of array **Lgrad** into the data file the elements of **Lgradv**(*ngradv*, 3) array are recorded line by line. Into each line of the file, the line number of the **Lgradv** array, number of a finite element and numbers of nodes of the element, in which the Neumann's boundary conditions for variable of vortex are given, are recorded. Behind of array **Lgradv** into the datafile the elements of the **Lpoint**(*nlp*) array are recorded line by line; into this elements are written the nodes numbers defining the output results. Into each line of the file the line number of the **Lpoint** array and also a node number of a finite element are recorded.

Behind of array **Lpoint** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the **x**-coordinates, into the *second* column of the **Coord** array the **y**-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (**x**, **y**)-coordinates are recorded.

Behind of array **Coord** the elements of **Bonds**(*nbsr*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions for variable of stream function, are recorded. Into each line of the datafile a line number of **Bonds** array and the Dirichlet's boundary condition are recorded. If such boundary conditions does not are preset, then one element of this array, namely **Bonds[1]=0**, is coded.

Behind of array **Bonds** the elements of **Bonds**(*nbsn*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions for variable of vortex, are recorded. Into each line of the datafile a line number of **Bonds** array and the Dirichlet's boundary condition are recorded. If such boundary conditions does not are preset, then one element of this array, namely **Bonds[1]=0**, is coded.

Behind of array **Bonds** the elements of **Grad**(*ngrad*, 2) array are recorded line by line. These elements define values of a *derivatives of stream function* over coordinates **x** and **y**. Into each line of the datafile a line number of **Grad** array and values of partial derivatives  $\frac{\partial \psi}{\partial x}$  and  $\frac{\partial \psi}{\partial y}$  are recorded.

At last, behind of array **Grad** the elements of **Grad**(*ngradv*, 2) array are recorded line by line. These elements define values of a *derivatives of vortex* over coordinates **x** and **y**. Into each line of the datafile a line number of **Grad** array and values of partial derivatives  $\frac{\partial \omega}{\partial x}$  and  $\frac{\partial \omega}{\partial y}$  are recorded.

In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbsr, nbsn, nwal, ngrad, ngradv, kcoord, nlpn, niteration, relax, tol, kprint, gama, reinold, ksolve, niter, toler, ntime, dtime*  
 Text line \*  
 Arrays **Mtop**(*nelem*, *nnode*)  
 Text line \*  
 Array **Lbdsr**(*nbsr*)  
 Text line \*  
 Array **Lbdsn**(*nbsn*)  
 Text line \*  
 Array **Mwall**(*nwal*, 3)  
 Text line \*  
 Array **Lgrad**(*ngrad*, 3)  
 Text line \*  
 Array **Lgradv**(*ngradv*, 3)  
 Text line \*  
 Array **Lpoint**(*nlp*)  
 Text line \*  
 Array **Ldebitas**(*ndeb*, 2)  
 Text line \*  
 Array **Coord**(*npoin*, *ndime*)

Text line \*  
 Array *Bonds*(*nbsr*)  
 Text line \*  
 Array *Bonds*(*nbsn*)  
 Text line \*  
 Array *Grad*(*ngrad*, 2)  
 Text line \*  
 Array *Grad*(*ngrad*, 2)

### 11.4.3. Brief description of the Vort\_Stream program solving the problem

The *Vort\_Stream* program was programmed on the *Maple*-language; it consists of the basic program and 43 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of a *stream function*, *vortex*, the *derivatives of vortex* in a time, projection of velocity of a motion of a fluid onto axis **X** or **Y** depending on parameter *kcoord*. The calculation results are output on the monitor and are recorded into a datafile, for that to variables *file\_rez1* and *file\_rez2* of the program must be ascribed qualifiers of the target files. Into the file *file\_rez1* values of a stream function and a vortex in nodes of finite elements, while into the file *file\_rez2* - values of a *velocity* in nodes, indicated in the array **Lpoint**, are recorded.

### 11.4.4. An example of use of the Maple-program Vort\_Stream

The current of a fluid in the closed square area of size 0.5x0.5, caused by motion of a upper bound with velocity  $u_x=1$ , is considered (fig. 11.14). The *boundary conditions* of the considered problem are represented also on the fig. 11.14.

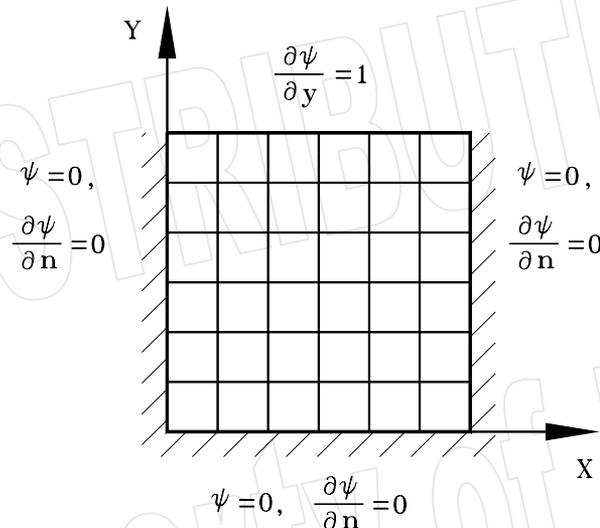


Fig. 11.14. The closed square area with a fluid and boundary conditions

Input data for the test example: the considered square area is divided onto *nelem*=36 finite elements, number of nodes *npoin*=49, number of boundary conditions of the Dirichlet on a free side *nbsr*=1, number of the sides of finite elements on the fixed sides *nwal*=18, number of boundary conditions of the Neumann for variable of stream function *ngrad*=6, number of boundary conditions of the Neumann for variable of a vortex *ngradv*=1, *kprint*=0, *ksolve*=1, *ngama*=0, *tol*=10<sup>(-6)</sup>, *niter*=50; *toler*=10<sup>(-6)</sup>, *niteration*=10, *relax*=0.5, Reynolds number *reinold*=20, the sizes of area 0.5x0.5, *kcoord*=1, *nlp*=7, *dtime*=0.2 sec.

Results of calculation over the test example:

The values of a stream function (**stream**), vortex (**vort**) and derivative of a vortex in a time (**dvort/dt**) in nodes (**node**) of finite elements are determined as follows:

node=1	stream=0	vort=0	dvort/dt=0
node=2	stream=0	vort=1.313331	dvort/dt=0
node=3	stream=0	vort=2.274448	dvort/dt=0
node=4	stream=0	vort=2.626065	dvort/dt=0
node=5	stream=0	vort=2.274448	dvort/dt=0
node=6	stream=0	vort=1.313287	dvort/dt=0
node=7	stream=0	vort=0	dvort/dt=0
node=8	stream=0	vort=1.313331	dvort/dt=0
node=9	stream=.004560	vort=11.218006	dvort/dt=79.057697
node=10	stream=.007897	vort=10.628012	dvort/dt=60.937568
node=11	stream=.009118	vort=10.782337	dvort/dt=59.261356
node=12	stream=.007897	vort=10.191322	dvort/dt=58.777694
node=13	stream=.004559	vort=10.413466	dvort/dt=73.555701
node=14	stream=0	vort=1.313382	dvort/dt=0
node=15	stream=0	vort=3.014351	dvort/dt=0
node=16	stream=.010466	vort=17.788264	dvort/dt=116.419841
node=17	stream=.018118	vort=9.512867	dvort/dt=17.468945
node=18	stream=.020916	vort=6.373283	dvort/dt=-15.810904
node=19	stream=.018118	vort=6.499613	dvort/dt=-5.156388
node=20	stream=.010466	vort=13.568930	dvort/dt=82.836176
node=21	stream=0	vort=3.014469	dvort/dt=0
node=22	stream=0	vort=5.610945	dvort/dt=0
node=23	stream=.019482	vort=34.979270	dvort/dt=246.432140
node=24	stream=.033671	vort=17.892760	dvort/dt=56.263208
node=25	stream=.038832	vort=9.533905	dvort/dt=-24.972443
node=26	stream=.033671	vort=8.118269	dvort/dt=-23.263606
node=27	stream=.019481	vort=21.584377	dvort/dt=134.002169
node=28	stream=0	vort=5.611165	dvort/dt=0
node=29	stream=0	vort=9.935256	dvort/dt=0
node=30	stream=.034497	vort=70.567367	dvort/dt=540.410512
node=31	stream=.059104	vort=37.861610	dvort/dt=197.806137
node=32	stream=.067953	vort=17.843068	dvort/dt=10.223405
node=33	stream=.059104	vort=11.680253	dvort/dt=-27.440893
node=34	stream=.034496	vort=33.020948	dvort/dt=212.028507
node=35	stream=0	vort=9.935652	dvort/dt=0
node=36	stream=0	vort=17.694621	dvort/dt=0
node=37	stream=.061439	vort=140.229248	dvort/dt=1152.379914
node=38	stream=.102291	vort=65.874205	dvort/dt=422.192432
node=39	stream=.114678	vort=29.755320	dvort/dt=85.443093
node=40	stream=.102291	vort=14.224394	dvort/dt=-38.542111

node=41	stream=.061437	vort=45.249118	dvort/dt=292.139432
node=42	stream=0	vort=17.695358	dvort/dt=0
node=43	stream=0	vort=35.938247	dvort/dt=0
node=44	stream=.124784	vort=182.434266	dvort/dt=1529.987856
node=45	stream=.170827	vort=81.040041	dvort/dt=549.830546
node=46	stream=.185019	vort=35.690775	dvort/dt=126.490128
node=47	stream=.170827	vort=18.695686	dvort/dt=-13.315544
node=48	stream=.124781	vort=28.545426	dvort/dt=124.630996
node=49	stream=0	vort=35.940036	dvort/dt=0

The values of velocity in nodes, indicated in the array **Lpoint(nlp)**, accept the form:

node=4	velx=.027606	vely=-.498154
node=11	velx=.027606	vely=-.498154
node=18	velx=.046169	vely=-.570500
node=25	velx=.091708	vely=-.647974
node=32	velx=.209493	vely=-.697937
node=39	velx=.639185	vely=-.350573
node=46	velx=.822466	vely=-.170314

The allocation of velocity  $u_x(x=0.25, y)$  along a vertical is represented on the fig. 11.15.

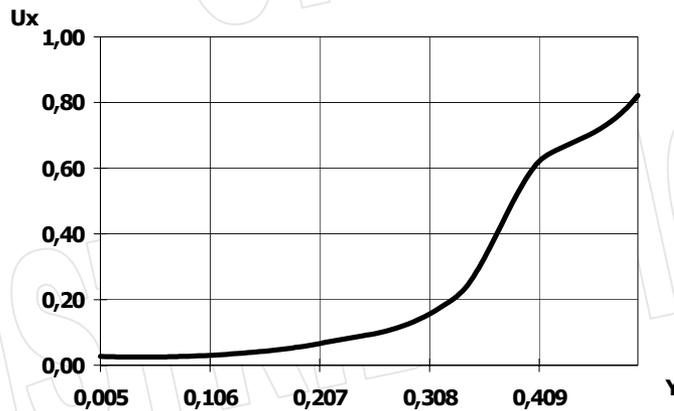


Fig. 11.15. Allocation of velocity  $u_x(x=0.25, y)$  along a vertical

The allocation of a stream function  $\psi$  in volume of a fluid is represented by isolines on fig. 11.16.

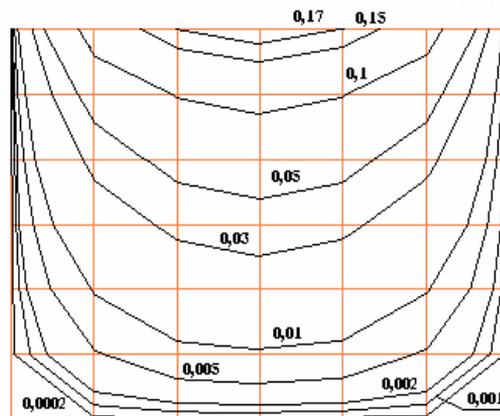


Fig. 11.16. Isolines of the stream function  $\psi$  in volume of a fluid

The *Vort\_Stream* program is intended for the solution of the Navier-Stokes's equations written in variables of *vortex-stream function* in cartesian or cylindrical axials for a case of an axial-symmetrical problem. With magnification of variables number, the time of calculation will increase, therefore Reynolds number should be selected rather small. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of attached to the present book in files *Mb4\_4\_new.mws*, *Mb4\_4.dat* and *Mb4\_4\_1.rez* accordingly.

### 11.5. Non-stationary motion of compressible fluid, described by potential of velocity

The non-stationary nonvortical motion of a compressible fluid, which approximately simulates some actual motions varying in a time, is considered. Such model allows with certain precision to solve many engineering problems having important appendices.

#### 11.5.1. The calculated equations of non-stationary potential motion of a fluid

The basic equation of a non-stationary motion of a compressible fluid, written by variable of potential of velocity, has the following form [70]:

$$c_0^2 \nabla^2 \varphi = \frac{\partial^2 \varphi}{\partial t^2} + (k - 1) \frac{\partial \varphi}{\partial t} \nabla^2 \varphi + 2 \nabla \varphi \frac{\partial}{\partial t} (\nabla \varphi) \tag{11.89}$$

where  $\varphi$  - potential of velocity;  $\nabla^2 \varphi$  - Laplace's operator ;  $c_0$  - velocity of a sound in a resting fluid;  $k$  - isentropic exponent. The one-dimensional motion of a fluid is considered. The equation (11.89) is solved by means of the FEM, using a quadratic one-dimensional isoparametric finite element represented on the *fig. 11.17*.

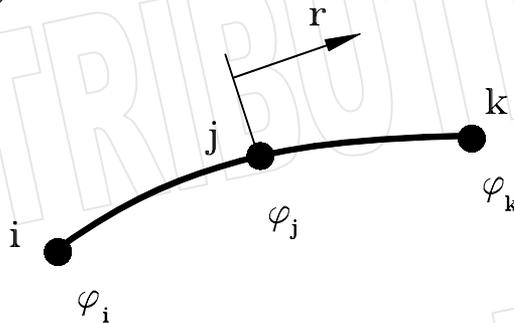


Fig. 11.17. A quadratic one-dimensional isoparametric finite element

Potential of velocity in limits of a finite element is approximated by the next expression:

$$\varphi(r, t) = \sum_{i=1}^3 N_i(r) \varphi_i(t) = [N]\{\varphi\}, \tag{11.90}$$

where  $[N(r)] = [N_1(r), N_2(r), N_3(r),]$ ;  $N_1(r) = \frac{1}{2}r(r - 1)$ ;  $N_2(r) = 1 - r^2$ ;  $N_3(r) = \frac{1}{2}r(1 + r)$  (11.91)

By substituting expression (11.90) into the equation (11.89) and by applying the Bubnov-Galerkin's procedure with integration by parts, we shall receive the following matrix equation for a finite element:

$$[m^{(e)}]\{\ddot{\phi}\} + [k^{(e)}]\{\phi\} = \{f^{(e)}(\phi, \dot{\phi})\}, \quad (11.92)$$

where  $[m^{(e)}] = \int_{L^{(e)}} A(x)[N]^T [N] dx$ ;  $[k^{(e)}] = c_0^2 \int_{L^{(e)}} A(x) \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] dx$ ;

$$\begin{aligned} \{f(\phi, \dot{\phi})\} &= \{f_1^{(e)}\} + \{f_2^{(e)}\} + \{f_3^{(e)}\} + \{f_4^{(e)}\}; \quad \{f_1^{(e)}\} = \frac{c_0^2 [N]^T \frac{\partial \phi}{\partial x} A(x)}{L}; \\ \{f_2^{(e)}\} &= \frac{(k-1)[N]^T \frac{\partial \phi}{\partial x} \frac{\partial \phi}{\partial t} A(x)}{L}; \quad \{f_3^{(e)}\} = (k-1) \int_{L^{(e)}} \left[ \frac{\partial N}{\partial x} \right]^T \left( \left[ \frac{\partial N}{\partial x} \right] \{\phi\} \right) ([N]\{\dot{\phi}\}) A(x) dx; \\ \{f_4^{(e)}\} &= 2 \int_{L^{(e)}} [N]^T \left( \left[ \frac{\partial N}{\partial x} \right] \{\phi\} \right) \left( \left[ \frac{\partial N}{\partial x} \right] \{\dot{\phi}\} \right) A(x) dx \end{aligned} \quad (11.93)$$

where  $A(x)$  – cross-sectional area of a pipeline. Using calculated relations (11.92), the global equations system of the following view is formed:

$$[M]\{\ddot{\Phi}\} + [K]\{\Phi\} = \{F\} \quad (11.94)$$

where  $[M]$ ,  $[K]$ ,  $\{F\}$  – global matrixes and vector. At numerical calculation of non-stationary motions of a compressible fluid in *acoustic approximation* the addend (11.93) can be neglected, what transmutes the equations system into a *linear*.

The equations system (11.94) is solved by the method of the off-center differences of the third order (*the Habol't's method*). In this method the cubic approximation of components of a vector on a left-side off-center difference template is used [71], namely:

$$\{\Phi(t)\} = \sum_{i=0}^3 \{A_i\} t^i \quad (11.95)$$

which reduces to a dependence of the following form:

$$\begin{aligned} \{\Phi(t)\} &= \{\Phi_t\} + \frac{1}{6\tau} (11\{\Phi_t\} - 18\{\Phi_{t-\tau}\} - 9\{\Phi_{t-2\tau}\} - \{\Phi_{t-3\tau}\}) + \\ &+ \frac{t^2}{2\tau^2} (2\{\Phi_t\} - 5\{\Phi_{t-\tau}\} + 4\{\Phi_{t-2\tau}\} - \{\Phi_{t-3\tau}\}) + \\ &+ \frac{t^3}{6\tau^3} (\{\Phi_t\} - 3\{\Phi_{t-\tau}\} + 3\{\Phi_{t-2\tau}\} - \{\Phi_{t-3\tau}\}), \end{aligned} \quad (11.96)$$

which, in turn, defines change of the vector components in an interval  $[t - 3*\tau, t]$ . In view of the dependence (11.96), the expressions for definition of velocities and accelerations at a moment  $t$  can be written as follows:

$$\{\dot{\Phi}(t)\} = \frac{1}{6\tau} (11\{\Phi_t\} - 18\{\Phi_{t-\tau}\} + 9\{\Phi_{t-2\tau}\} - 2\{\Phi_{t-3\tau}\}); \quad (11.97)$$

$$\{\ddot{\Phi}(t)\} = \frac{1}{\tau^2} (2\{\Phi_t\} - 5\{\Phi_{t-\tau}\} + 4\{\Phi_{t-2\tau}\} - \{\Phi_{t-3\tau}\}) \quad (11.98)$$

where:  $\tau$  - integration step. The expressions (11.97), (11.98) allow to receive from the equations (11.94) a recurring dependence of the following form:

$$\left(\frac{2}{\tau}[\mathbf{M}] + [\mathbf{K}]\right)\{\Phi_t\} = \{\mathbf{F}_t\} + \frac{1}{\tau^2} [\mathbf{M}](5\{\Phi_{t-\tau}\} - 4\{\Phi_{t-2\tau}\} + \{\Phi_{t-3\tau}\}) \quad (11.99)$$

$$\text{or } [\mathbf{A}]\{\Phi_t\} = \{\mathbf{Q}_t\} \quad (11.100)$$

The contour values in a time, on the first and the second step are defined by the following relations:

$$\{\Phi_{-2\tau}\} = 9\{\Phi_0\} - 8\{\Phi_\tau\} + 6\tau\{\dot{\Phi}_0\} + 6\tau^2\{\ddot{\Phi}_0\}; \quad (11.101)$$

$$\{\Phi_{-\tau}\} = 2\{\Phi_0\} - \{\Phi_\tau\} + \tau^2\{\ddot{\Phi}_0\} \quad (11.102)$$

The vectors  $\{\Phi_0\}$ ,  $\{\dot{\Phi}_0\}$  are determined from the initial conditions, while the vector  $\{\ddot{\Phi}_0\}$  - from the solution of the equations system of the following form:

$$[\mathbf{M}]\{\ddot{\Phi}_0\} = \{\mathbf{F}_0\} - [\mathbf{K}]\{\Phi_0\} \quad (11.103)$$

The given dependences change a little the matrix  $[\mathbf{A}]$  and vector  $\{\mathbf{Q}\}$  on the *first* step of integration, namely:

$$[\mathbf{A}] = \frac{6}{\tau^2} [\mathbf{M}] + [\mathbf{K}]; \quad (11.104)$$

$$\{\mathbf{Q}_t\} = \{\mathbf{F}_t\} + \frac{1}{\tau^2} [\mathbf{M}](6\{\Phi_{t-\tau}\} + 6\tau\{\dot{\Phi}_0\} + 2\tau^2\{\ddot{\Phi}_0\}) \quad (11.105)$$

and also the vector  $\{\mathbf{Q}\}$  on the *second* integration step, namely:

$$\{\mathbf{Q}_t\} = \{\mathbf{F}_t\} + \frac{1}{\tau^2} [\mathbf{M}](4\{\Phi_{t-\tau}\} - 2\{\Phi_0\} + \tau^2\{\ddot{\Phi}_0\}) \quad (11.106)$$

### 11.5.2. Input data for the solution of the problem

For the solution of the equation describing a motion of a *compressible fluid*, the *Dpot\_flow* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary *qualifier* of the file is ascribed to variable  $\mathbf{F}$ . The data in the file are placed in the strict order. In the *first* line the number of finite elements (parameter *nelem*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions (*nbond*), the number of Neumann's boundary conditions (*ngrad*), and the number defining through what number of integration steps the computation results (*nstep*) are written, are coded.

In the *second* line, a print code of intermediate results (*kprint*), a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*) are coded. If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0; otherwise - in *cylindrical* frames an axial-symmetric problem is solved. if parameter *ksolve*=0, then for the solution, the *solve*-function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In the *third* line of the datafile the following parameters are recorded: the number of integration

steps in a time (*ntime*), an integration step (*dtime*), velocity of a sound in a resting fluid (*greitis*), density of a fluid (*tankis*) and an initial pressure in a pipeline (*pressure0*).

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode* =3} are coded. After of array **Mtop** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into the **Lbond** array the numbers of nodes, in which are known Dirichlet's boundary conditions, are coded.

Behind of array **Lbond** into the data file the elements of **Lgrad**(*ngrad*, 2) array are recorded line by line. Into each line of the file the line number and elements of **Lgrad** array are recorded. Into the *first* column of the **Lgrad** array the number of a finite element, while into the *second* the number of node of the element on which the Neumann's boundary conditions are known, are recorded.

Behind of array **Lgrad** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=1), are recorded line by line. Into each line of the file the number of a node and its *x*-coordinate are coded. Behind of array **Coord** the elements of **Splotas**(*nelem*, *nmode*) array are recorded line by line. Into this array the areas of the cross-sections of a pipeline with a fluid are recorded. Into each line are introduced a line number of the array **Splotas** and three values of a cross-sectional area of the pipeline.

Behind of array **Splotas** the elements of **Bond**(*nbond*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions, i.e. the values of stream function, are recorded. Into each line of the datafile a line number of **Bond** array and a boundary value of stream function are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

In each subsequent *ngrad* lines of the datafile, the elements of the **Grad**(*ngrad*, 4) array are recorded line by line. Into each line of this array the values of coefficients **a<sub>0</sub>**, **a<sub>1</sub>**, *alfa* and *beta*, which define derivative of potential of velocity (Neumann's boundary condition, i.e.  $\frac{\partial\phi(t)}{\partial x} = (a_0 + a_1 \cos(\text{alfa} \cdot t)) e^{\text{beta} \cdot t}$ ) are recorded. Into each line of the datafile a line number of **Grad** array and four its elements are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbond, ngrad, nstep, kprint, ksolve, niter, toler, ntime, dtime, greitis, tankis, pressure0*  
Text line \*  
Arrays **Mtop**(*nelem*, *nmode*)  
Text line \*  
Array **Lbond**(*nbond*)  
Text line \*  
Array **Lgrad**(*ngrad*, 3)  
Text line \*  
Array **Coord**(*npoin*, *ndime*)  
Text line \*  
Array **Splotas**(*nelem*, *nmode*)  
Text line \*  
Array **Bond**(*nbond*)  
Text line \*  
Array **Grad**(*ngrad*, 4)

### 11.5.3. Brief description of the Dpot\_flow program solving the problem

The *Dpot\_flow* program was programmed on the *Maple*-language; it consists of the basic program and 27 procedures. All procedures can be divided into three groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of potential of velocity, the first and the second derivatives in a time, a velocity and also hydrodynamic pressure in nodes of the finite elements. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3*, *file\_1*, *file\_2* and *file\_3* of the program must be ascribed qualifiers of the target files. Into the files *file\_rez1* and *file\_rez2* values of *potential of velocity*, its *first* and the *second* derivatives in a time, and also velocities in nodes of finite elements are recorded accordingly. Into the files *file\_rez3* values of pressure in nodes of finite elements are recorded. Into the files *file\_1*, *file\_2* and *file\_3* values of velocity, pressure and potential of velocity in each node of a finite element on each integration step *nstep* are recorded accordingly.

### 11.5.4. An example of use of the Maple-program Dpot\_flow

The motion of a fluid in a pipeline of the composite cross-section is considered (fig. 11.18).

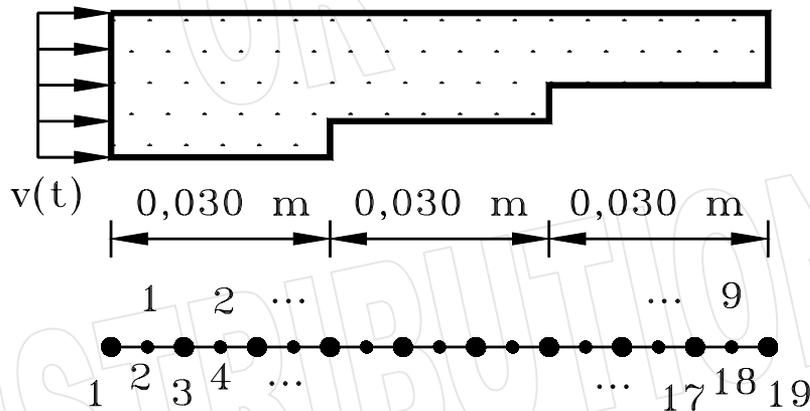


Fig. 11.18. The calculated scheme of a pipeline

On the left-end of the pipeline the velocity of a fluid  $v(t)=10$  m/s, and on the right-end - boundary conditions  $\varphi_r = 0$  are preset.

Input data for the test example: the pipeline with fluid is divided onto  $nelem=9$  one-dimensional quadratic finite elements, number of nodes  $npoin=19$ , number of boundary conditions of Dirichlet  $nbond=1$ , number of boundary conditions of the Neumann  $ngrad=1$  at  $nstep=10$ ,  $kprint=0$ ,  $ksolve = 0$ ,  $ntime=4000$ ,  $dtime=5.0 \cdot 10^{-4}$  sec,  $greitis=10^3$  m/s,  $tankis=10^3$  kg/m<sup>3</sup>,  $pressure0 = 10^5$  Pa.

#### Results of calculation over the test example:

The values of potential of velocity and its first and the second derivatives after  $ntime$  integration steps accept the following form accordingly:

node=1	105.811088	-3.333333e-10	0
node=2	-15.426106	-1.333333e-10	0
node=3	18.081336	6.666666e-11	0

node=4	-2.638178	-1.333333e-11	2.8e-07
node=5	3.106792	3.333333e-11	-2e-07
node=6	-.465684	-6.666666e-13	-2.4e-08
node=7	.633279	-6.666666e-12	-2e-08
node=8	-.092326	-2e-12	0
node=9	.108228	0	0
node=10	-.015799	1.333333e-13	0
node=11	.018666	-6.666666e-14	0
node=12	-.002848	-1.333333e-14	8e-11
node=13	.004211	0	0
node=14	-.000613	0	-4e-12
node=15	.000718	-1.666666e-14	2.4e-11
node=16	-.000104	1e-15	0
node=17	.000119	6.666666e-16	0
node=18	-1.486e-05	6.	0

The values of velocity in nodes of finite elements after ntime integration steps accept the following form accordingly:

ie=1	9.930475	2.193243	-5.543988
ie=2	1.697587	.374363	-.948860
ie=3	.295409	.061837	-.171734
ie=4	.059434	.013126	-.033181
ie=5	.010163	.002239	-.005685
ie=6	.001790	.000361	-.001067
ie=7	.000395	8.730731e-05	-.000220
ie=8	6.733911e-05	1.498909e-05	-3.736091e-05
ie=9	1.043689e-05	2.983633e-06	-4.469627e-06

The values of pressure in nodes of finite elements after ntime integration steps accept the following form accordingly:

ie=1	50692.826483	97594.840803	84632.098209
ie=2	98559.097664	99929.925949	99549.831679
ie=3	99956.366503	99998.088041	99985.253677
ie=4	99998.233777	99999.913850	99999.449482
ie=5	99999.948348	99999.997493	99999.983836
ie=6	99999.998397	99999.999934	99999.999430
ie=7	99999.999921	99999.999996	99999.999975
ie=8	99999.999997	99999.999999	99999.999999
ie=9	99999.999999	99999.999999	99999.999999

On the fig. 11.19 the change of velocity in nodes of finite elements at a moment  $t=2$  s is represented.

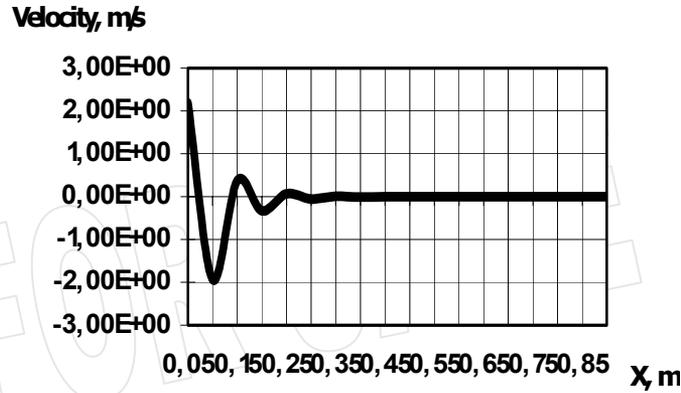


Fig. 11.19. Dependence of the change of velocity in nodes of finite elements at a moment  $t=2$  s

On the fig. 11.20 the change of pressure in nodes of finite elements at a moment  $t=2$  s is represented.

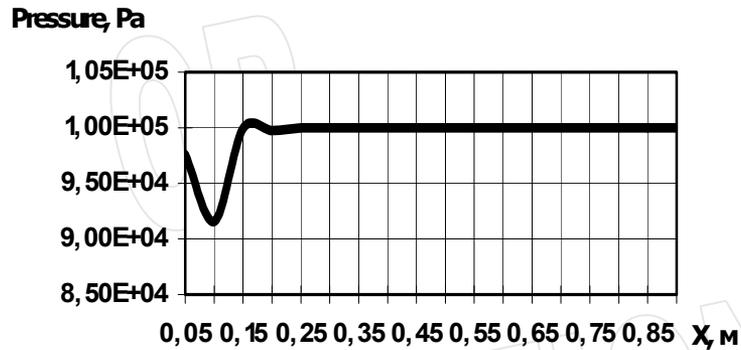


Fig. 11.20. Dependence of the pressure of velocity in nodes of finite elements at a moment  $t=2$  s

The *Dplot\_flow* program is intended for the solution of an equation written for *variable of potential of velocity* and describing a non-stationary motion of a compressible fluid in a pipeline with a variable cross-section. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in files *Mb4\_5\_new.mws*, *Mb4\_5.dat* and *Mb4\_5\_1.rez* accordingly.

# Chapter 12.

## Some applied problems of hydromechanics

The investigation of process of a motion of a fluid and gas constitutes the important problem not only in various fields of engineering, but also in biology, medicine, in many other fields of the natural and scientific appendices. Each physical process happening in a concrete system, has own features, which are characteristic for this process. These features can depend on geometrical parameters of system, properties of a fluid and gas, of action of external medium, of duration of process etc. In the given chapter some applied problems of hydromechanics are considered such as: hydrodynamic lubrication between moving surfaces; motion of a fluid with a nonlinear dependence of viscosity; convective heat exchange and mass transfer; transitional process of a motion of a fluid in a hydraulic system. These problems are successfully solved in medium of the mathematical package *Maple*.

### 12.1. Equation of Reynolds for a layer of lubrication

Friction surfaces of details of machines are parted by a lamina of a viscous fluid or gas, in which the pressure preventing touch of surfaces develops. The regularities of a motion of such lamina are described by the theory of hydrodynamic lubrication, the theoretical basics of which are based on works of O.Reynolds, N.P. Petrov, N.E. Zhukovsky, A.S. Chaplygin [72]. One of the basic features of motion of layer of lubrication is the smallness of its thickness (*film thickness*) in comparison with the sizes of adjoining surfaces.

#### 12.1.1. The calculated equations of Reynolds

The layer of a fluid between approximately parallel surfaces is considered (*fig. 12.1*). The motion is considered steady-state and the activity of mass forces is supposed inessential.

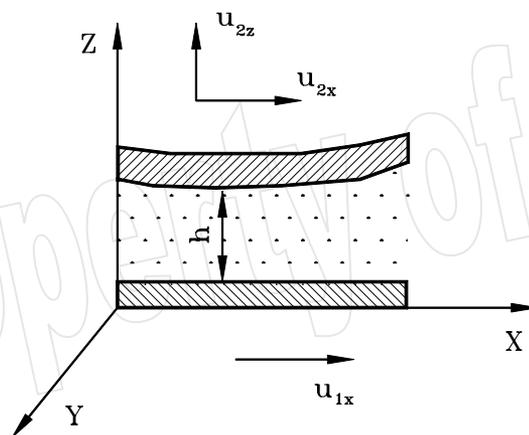


Fig. 12.1. The scheme of motion of a fluid between nonparallel solid surfaces

The  $X$  axis lies on a lower surface, which moves with velocity  $u_{1x}$ . The second surface can be motionless or can move with velocity  $u_{2x}$  along axis  $X$  and with velocity  $u_{2z}$  along axis  $Z$ . If during a motion the thickness  $h$  of a layer is remaining a rather small, then the relation of velocities  $\frac{u_{2z}}{u_{2x}}$  also should be small. Then for any point of lubrication the following relation  $u_x \gg u_z$  is fulfilled. The change of velocity in direction of axis  $Z$ , owing to a smallness of a layer, happens more intensively, than along axis  $X$ , i.e. for any component of velocity  $u_i$  are valid the following relations:

$$\frac{\partial u_i}{\partial z} \gg \frac{\partial u_i}{\partial x}; \quad \frac{\partial u_i}{\partial z} \gg \frac{\partial u_i}{\partial y}; \quad \frac{\partial^2 u_i}{\partial z^2} \gg \frac{\partial^2 u_i}{\partial x^2}; \quad \frac{\partial^2 u_i}{\partial z^2} \gg \frac{\partial^2 u_i}{\partial y^2} \quad (12.1)$$

These reasons allow to reject in equations of motion not only inertial terms, but also those viscosity terms, which contain derivatives over  $X$  and  $Y$ . Neglecting the small terms, instead of the equation of Navier-Stokes the equations system of a motion of the following form are gained:

$$\frac{\partial p}{\partial x} = \mu \frac{\partial^2 u_x}{\partial z^2}; \quad \frac{\partial p}{\partial z} = 0; \quad \frac{\partial p}{\partial y} = \mu \frac{\partial^2 u_y}{\partial z^2}; \quad \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0. \quad (12.2)$$

The equations system (12.2) are called as Reynolds equations system for a layer of lubrication. This equations system can be used for the solution of manifold problems linked with a motion of a layer of lubrication. The equations of Reynolds are obtained by a deletion not only of all inertial terms, but also part of viscosity terms. Therefore, given equations, in comparison with the equations of Navier-Stokes, describe a process of a motion less detailly. The Reynolds equation in case of two-dimensional allocation of pressure in a layer of lubrication accepts the following form:

$$\frac{\partial}{\partial x} \left( \frac{h^3}{12\mu} \frac{\partial p}{\partial x} \right) + \frac{\partial}{\partial z} \left( \frac{h^3}{12\mu} \frac{\partial p}{\partial y} \right) = \frac{\partial}{\partial x} \left( h u_x + \frac{h^3}{12\mu} F_x \right) + \frac{\partial}{\partial y} \left( h u_y + \frac{h^3}{12\mu} F_y \right) + \frac{\partial h}{\partial t} + u_d \quad (12.3)$$

where:  $u_x, u_y$  - velocities of motion of a fluid in a layer in direction of axes  $X$  and  $Y$  accordingly;  $F_x, F_y$  - external forces in direction of axes  $X$  and  $Y$  accordingly;  $\frac{\partial h}{\partial t}$  - velocity of change of thickness of a layer of lubrication;  $u_d$  - velocity of a diffusion stream via surfaces ( $u_d > 0$ , if the velocity of a diffusion stream is directed out of a layer of lubrication;  $u_d < 0$ , if the velocity of a diffusion stream is directed into a layer of lubrication). The equation (12.3) is solved by the Rayleigh-Ritz's method. The functional corresponding to the given equation, accepts the following form:

$$J(p) = \int_A \left\{ \frac{h^3}{24\mu} \left[ \left( \frac{\partial p}{\partial x} \right)^2 + \left( \frac{\partial p}{\partial y} \right)^2 \right] - h \left( u_x \frac{\partial p}{\partial x} + u_y \frac{\partial p}{\partial y} \right) + \frac{h^3}{12\mu} \left( F_x \frac{\partial p}{\partial x} + F_y \frac{\partial p}{\partial y} \right) + p \left( \frac{\partial h}{\partial t} + u_d \right) \right\} dA + \int_{\Gamma} q p d\Gamma \quad (12.4)$$

where  $q$  - stream of a fluid, normal to a surface ( $q > 0$ , if the stream of a fluid comes out from a layer of lubrication;  $q < 0$ , if the stream of a fluid comes in a layer of lubrication). For obtaining of the solution the two-dimensional nine-nodal quadratic isoparametric finite element, represented on the fig. 12.2, is used.

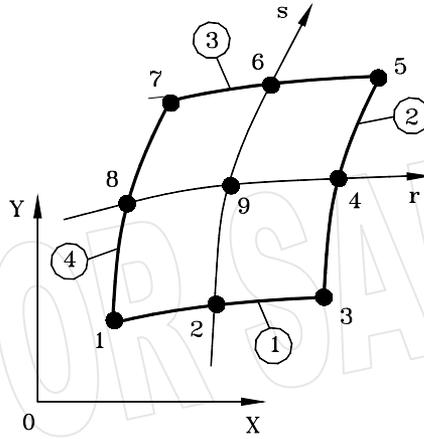


Fig. 12.2. Two-dimensional nine-nodal quadratic isoparametric finite element

The pressure in such finite element is approximated by the next relations:

$$p(r, s) = [N(r, s)]\{p\}, \quad (12.5)$$

$$\text{where } N_1(r, s) = \frac{1}{4}(1-r)(1-s) - \frac{1}{2}N_2(r, s) - \frac{1}{2}N_8(r, s) - \frac{1}{4}N_9(r, s);$$

$$N_2(r, s) = \frac{1}{2}(1-r^2)(1-s) - \frac{1}{2}N_9(r, s); \quad N_3(r, s) = \frac{1}{4}(1+r)(1-s) - \frac{1}{2}N_2(r, s) - \frac{1}{2}N_4(r, s) - \frac{1}{4}N_9(r, s);$$

$$N_4(r, s) = \frac{1}{2}(1-s^2)(1+r) - \frac{1}{2}N_9(r, s); \quad (12.6)$$

$$N_5(r, s) = \frac{1}{4}(1+r)(1+s) - \frac{1}{2}N_4(r, s) - \frac{1}{2}N_6(r, s) - \frac{1}{4}N_9(r, s); \quad N_6(r, s) = \frac{1}{2}(1-r^2)(1+s) - \frac{1}{2}N_9(r, s);$$

$$N_7(r, s) = \frac{1}{2}(1-r^2)(1+s) - \frac{1}{2}N_9(r, s); \quad N_8(r, s) = \frac{1}{2}(1-s^2)(1-r) - \frac{1}{2}N_9(r, s); \quad N_9(r, s) = (1-r^2)(1-s^2)$$

Then the equations system for a finite element  $e$  accepts the following form:

$$[k^{(e)}]\{p\} = \{f^{(e)}\}, \quad (12.7)$$

$$\text{where } [k^{(e)}] = \int_{A^{(e)}} \frac{h^3}{12\mu} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA; \quad \{f^{(e)}\} = \{f_u^{(e)}\} + \{f_f^{(e)}\} - \{f_h^{(e)}\} - \{f_b^{(e)}\} - \{f_q^{(e)}\};$$

$$\{f_u^{(e)}\} = \int_{A^{(e)}} h(x, y) \left( \left[ \frac{\partial N}{\partial x} \right]^T u_x + \left[ \frac{\partial N}{\partial y} \right]^T u_y \right) dA; \quad \{f_q^{(e)}\} = \int_{A^{(e)}} [N]^T q dA \quad (12.8)$$

$$\{f_f^{(e)}\} = \int_{A^{(e)}} \frac{h^3(x, y)}{12\mu} \left( \left[ \frac{\partial N}{\partial x} \right]^T F_x + \left[ \frac{\partial N}{\partial y} \right]^T F_y \right) dA; \quad \{f_h^{(e)}\} = \int_{A^{(e)}} [N]^T \frac{\partial h}{\partial t} dA; \quad \{f_d^{(e)}\} = \int_{A^{(e)}} [N]^T u_d dA;$$

The *thickness* of a layer of lubrication in each finite element is approximated by the next expression:

$$h(\mathbf{r}, \mathbf{s}) = [\mathbf{N}(\mathbf{r}, \mathbf{s})] \{h^{(e)}\} \quad (12.9)$$

The *common force of pressure* is defined by the following relation:

$$\mathbf{F} = \sum_{e=1}^{NE} \int_{A^{(e)}} p^{(e)}(x, y) \, dA \quad (12.10)$$

### 12.1.2. Input data for the solution of the problem

For the solution of the Reynolds equation for a *layer of lubrication*, the *Reynold* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. In the *first* line the next parameters are coded: the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions (*nbond*), number of finite elements in which the stream of a fluid (*nbq*) is preset, number of finite elements in which the velocity of a diffusion (*nbd*) is preset. In the *second* line the next parameters are coded: a print code of intermediate results (*kprint*), velocity of a surface in direction of axes **X** and **Y** (*velocx* and *velocity*), values of forces acting in direction of axes **X** and **Y** (*bodyx* and *bodyy*), velocity of change of thickness of a layer (*hvel*), and dynamic viscosity of a fluid (*amin*). If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out.

In the *third* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). if parameter *ksolve*=0, then for the solution, the *'solve'* function of the Maple-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element (*array Mtop(nelem, nnode)*, where *nnode* – number of nodes of the finite element, i.e. *nnode* =9) are coded. After of array **Mtop** the elements of **Lbond**(*nbond*) array are coded line by line. Into each line of the file the line number and element of **Lbond** array are recorded. Into the **Lbond** array the numbers of nodes, in which are known Dirichlet's boundary conditions, are coded.

After of array **Lbond** the elements of **Lbq**(*nbq*) array are coded line by line. Into each line of the file are recorded the line number of the array **Lbq** and also the number of a finite element, in which is known a stream of a fluid **q**, coming in or coming out from a layer of lubrication. After of array **Lbq** the elements of **Lbd**(*nbd*) array are coded line by line. Into each line of the file are recorded the line number of the array **Lbd** and also the number of a finite element, in which the velocity of diffusion of a fluid (*hvel*) is known.

Behind of array **Lbd** into the data file the elements of **Coord**(*npoint, ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the **x**-coordinates, into the *second* column of the **Coord** array the **y**-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (**x**, **y**)-coordinates are recorded. Behind of array **Coord** the elements of **H<sub>z</sub>**(*nelem, nmode*) array are coded line by line. Into each line are recorded the line number of the array **H<sub>z</sub>** and also *thickness* of a layer of lubrication in each node of a finite element.

Behind of array **H<sub>z</sub>** the elements of **Bond**(*nbond*) array are recorded line by line. Into this array the values of the Dirichlet's boundary conditions, i.e. the values of *pressure*, are recorded. Into each line of the datafile a line number of **Bond** array and a boundary value of *potential of velocity* are recorded. The lines of **Bond** array should correspond to the lines of **Lbond** array.

In subsequent *nbq* lines of the datafile, the elements of the **Bq**(*nbq*) array are recorded line by line. Into each line of the datafile are recorded the line number of the **Bq** array and also values of stream of a fluid **q**. Behind of array **Bq** the elements of **Bd**(*nb<sub>d</sub>*) array are recorded line by line. Into each line of the datafile are recorded the line number of the **Bd** array and also value of velocity of a diffusion via surface. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbond, nbq, nbd, kprint, velocx, velocity, bodyx, bodyy, hvel, amin, ksolve, niter, toler*  
Text line \*  
Arrays **Mtop**(*nelem, nnode*)  
Text line \*  
Array **Lbond**(*nbond*)  
Text line \*  
Array **Lbq**(*nbq*)  
Text line \*  
Array **Lbd**(*nbd*)  
Text line \*  
Array **Coord**(*npoin, ndime*)  
Text line \*  
Array **H<sub>z</sub>**(*nelem, nnode*)  
Text line \*  
Array **Bond**(*nbond*)  
Text line \*  
Array **Bq**(*nbq*)  
Text line \*  
Array **Bd**(*nbd*)

### 12.1.3. Brief description of the Reynold program solving the problem

The *Reynold* program was programmed on the *Maple*-language; it consists of the basic program and 19 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of pressure and total force of pressure in nodes of finite elements. The calculation results are output on the monitor and are recorded into datafiles, for that to variable *file\_rez1* must be ascribed qualifiers of the target files. In this file the values of pressure and total force of pressure are recorded.

### 12.1.4. An example of use of the Maple-program Reynold

As an applied aspect of the program, the flat motion of a fluid between two nonparallel planes is considered; the lower plane moves with a stationary velocity  $\mathbf{u}_x$  in direction of axis **X** (*fig. 12.3*).

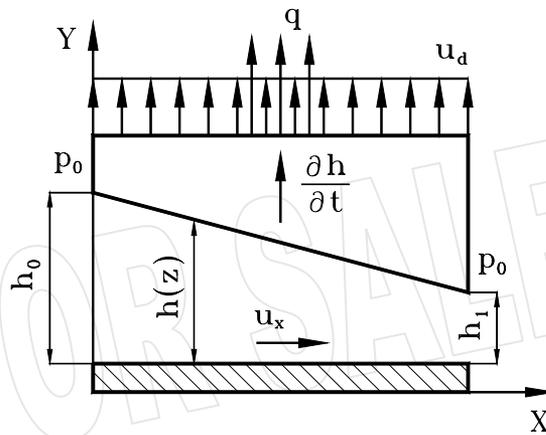


Fig. 12.3. Scheme of a flat cuneiform layer of lubrication

The space at the left and to the right of the plate is filled up by a viscous fluid, being under a pressure  $p_0$ . The plate can move with a stationary velocity  $\frac{\partial h}{\partial t}$  in direction of axis  $Z$ . Through the plate can pass a diffusion stream with velocity  $u_d$ . On some part of the plate the normal stream of a fluid  $q$  can be known. The calculated scheme of a cuneiform layer of lubrication is represented on the fig. 12.4.

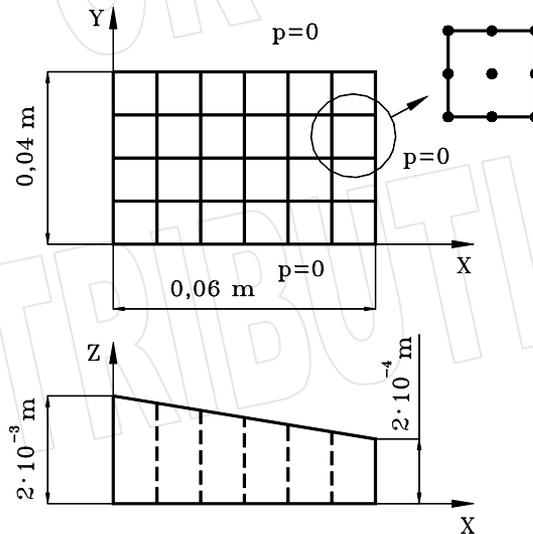


Fig. 12.4. The calculated scheme of cuneiform layer of lubrication

Input data for the test example:  $nelem=24$ ,  $npoin=117$ ,  $nbq=1$ ,  $nbd=1$ ,  $velocx=20$  m/s,  $velocity=0$ ,  $bodyx=0$ ,  $bodyy=0$ ,  $hvel=-2.0$  m/s,  $amin=10^{(-3)}$  N·s/m<sup>2</sup> (water at temperature 293 K),  $kprint=1$ ,  $ksolve=0$ ,  $niter=50$ ,  $toler=10^{(-6)}$ .

Results of calculation over the test example:

Values of pressures (**pressure**) in nodes (**node**) of the finite elements are:

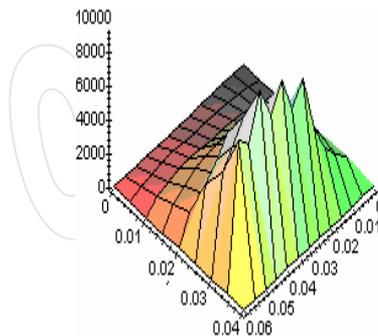
node=1	pressure=0e-01	node=6	pressure=0e-01
node=2	pressure=0e-01	node=7	pressure=0e-01
node=3	pressure=0e-01	node=8	pressure=0e-01
node=4	pressure=0e-01	node=9	pressure=0e-01
node=5	pressure=0e-01	node=10	pressure=0e-01

node=11	pressure=0e-01	node=54	pressure=0e-01
node=12	pressure=0e-01	node=55	pressure=2.244627e+03
node=13	pressure=0e-01	node=56	pressure=3.592984e+03
node=14	pressure=0e-01	node=57	pressure=4.526137e+03
node=15	pressure=3.188597e+02	node=58	pressure=5.555803e+03
node=16	pressure=4.598600e+02	node=59	pressure=4.949443e+03
node=17	pressure=5.255662e+02	node=60	pressure=0e-01
node=18	pressure=6.172883e+02	node=61	pressure=0e-01
node=19	pressure=4.820098e+02	node=62	pressure=1.468212e+03
node=20	pressure=0e-01	node=63	pressure=2.678836e+03
node=21	pressure=0e-01	node=64	pressure=3.499878e+03
node=22	pressure=3.185353e+02	node=65	pressure=4.372170e+03
node=23	pressure=5.924044e+02	node=66	pressure=4.901569e+03
node=24	pressure=7.680934e+02	node=67	pressure=5.382332e+03
node=25	pressure=8.611446e+02	node=68	pressure=5.958599e+03
node=26	pressure=9.086777e+02	node=69	pressure=6.865655e+03
node=27	pressure=9.666354e+02	node=70	pressure=7.860662e+03
node=28	pressure=1.032965e+03	node=71	pressure=6.680879e+03
node=29	pressure=1.108619e+03	node=72	pressure=5.691643e+03
node=30	pressure=1.126801e+03	node=73	pressure=0e-01
node=31	pressure=8.771808e+02	node=74	pressure=0e-01
node=32	pressure=6.218418e+02	node=75	pressure=2.240496e+03
node=33	pressure=0e-01	node=76	pressure=4.977942e+03
node=34	pressure=0e-01	node=77	pressure=7.344495e+03
node=35	pressure=9.905035e+02	node=78	pressure=8.228840e+03
node=36	pressure=1.631841e+03	node=79	pressure=7.652832e+03
node=37	pressure=2.086019e+03	node=80	pressure=0e-01
node=38	pressure=2.109412e+03	node=81	pressure=0e-01
node=39	pressure=1.527858e+03	node=82	pressure=0e-01
node=40	pressure=0e-01	node=83	pressure=0e-01
node=41	pressure=0e-01	node=84	pressure=0e-01
node=42	pressure=6.763882e+02	node=85	pressure=0e-01
node=43	pressure=1.308876e+03	node=86	pressure=0e-01
node=44	pressure=1.730693e+03	node=87	pressure=0e-01
node=45	pressure=2.237067e+03	node=88	pressure=0e-01
node=46	pressure=2.716554e+03	node=89	pressure=0e-01
node=47	pressure=2.987038e+03	node=90	pressure=0e-01
node=48	pressure=3.179612e+03	node=91	pressure=0e-01
node=49	pressure=2.993478e+03	node=92	pressure=0e-01
node=50	pressure=2.693639e+03	node=93	pressure=0e-01
node=51	pressure=2.088934e+03	node=94	pressure=1.832136e+02
node=52	pressure=1.530580e+03	node=95	pressure=4.139115e+02
node=53	pressure=0e-01	node=96	pressure=5.054916e+02

node=97	pressure=5.730083e+02	node=108	pressure=4.100065e+03
node=98	pressure=6.190663e+02	node=109	pressure=4.961157e+03
node=99	pressure=3.509228e+02	node=110	pressure=6.005858e+03
node=100	pressure=5.147851e+02	node=111	pressure=4.118054e+03
node=101	pressure=1.316712e+03	node=112	pressure=1.531019e+03
node=102	pressure=1.916201e+03	node=113	pressure=3.152675e+03
node=103	pressure=2.199646e+03	node=114	pressure=8.522706e+03
node=104	pressure=1.994115e+03	node=115	pressure=9.620758e+03
node=105	pressure=1.079669e+03	node=116	pressure=1.025503e+04
node=106	pressure=1.197451e+03	node=117	pressure=8.546528e+03
node=107	pressure=2.920710e+03		

The graph of allocation of pressure in a layer of lubrication is represented on the *fig. 12.5*.

**Pressure  $p(x,y)$  Distribution:**



**Fig. 12.5.** The graph of allocation of pressure in a layer of lubrication

The *Reynold* program is intended for the solution of two-dimensional Reynolds equation for a *layer of lubrication* in the cartesian frame. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb9\_1\_new.mws*, *Mb9\_1.dat* and *Mb9\_1\_1.rez* accordingly.

## 12.2. A model of motion of a non-Newtonian fluid

The important role in perfecting of machines and mechanisms, and also in rational use of existing constructions belongs to study of processes of *friction*, *lubrication* and *deterioration*. The hydrostatic liquid lubrication (*hydrostatic effect*) is provided by continuous forced feeding of a fluid from the pumping station into a backlash between surfaces of friction. The hydrostatic pairs of friction have received considerable distribution in mechanical engineering as the *bearings* and *directings*, *frontal obturatings* and *drives of small displacements*.

The efficiency of hydrostatic lubrication is defined by properties of the used lubricant fluids, and also by their influence onto the characteristics of friction nodes. The incremental requirements to modern machines and mechanisms caused expediency of use of fluids which are differed from the classical *Newtonian fluid*. *Non-Newtonian reological composite fluids* have new properties (*nonlinear viscosity*, *viscoplasticity*, *viscoelasticity*), what allows to receive qualitatively new effects in an acting layer of lubrication.

### 12.2.1. The calculated equations of motion of a non-Newtonian fluid

In the base of *classical hydrodynamics* lie the Newton's laws, according to which in case of *rectilinear schistose (laminar)* motion of a fluid the dependence between a tangential stress  $t$  acting in a plane of touch of layers of a fluid, and derivative of velocity in direction, normal to this plane  $\frac{\partial u}{\partial n}$  (*velocity of detrusion*), is defined by the following relation:

$$\tau = \mu \frac{\partial u}{\partial n} \quad (12.11)$$

where  $\mu$  – *coefficient of dynamic viscosity* of a fluid. The fluids satisfying to the relation (12.11), have received the title *Newtonian fluids*. Really, many materials, in particular, melts and mortars of polymetric compounds, suspensions, clay mortars, oil colours, pharmaceutical and foodstuff, blood etc., differ from the *Newtonian fluids*. The viscosity of such fluids is some function of velocity of detrusion and temperature. Moreover such fluids can reveal also *plastic properties*, when at some longitudinal stress of detrusion takes place a *fluidity*. Furthermore, these materials can reveal also *viscoelastic properties*, when the state of medium is defined by a *history of its deforming*. Motion of a non-linear viscous fluid is considered below, the *state equation* of which is defined by the following relation:

$$\sigma_{ij} = -p\delta_{ij} + \varphi(l_2) \dot{\epsilon}_{ij} \quad (12.12)$$

where:  $\sigma_{ij}$ ,  $\delta_{ij}$ ,  $\dot{\epsilon}_{ij}$  – *components of stresses tensors, unit tensor and velocities of strain* accordingly;  $\varphi$  – *viscosity* depending from  $l_2$  – of the *second invariant of tensor of velocities of strain*;  $p$  – *pressure*. The *second invariant of tensor of velocities of strain* accepts the following form:

$$l_2 = \sum_{i=1}^3 \sum_{j=1}^3 \dot{\epsilon}_{ij} \dot{\epsilon}_{ji} \quad (12.13)$$

While the components of tensor of velocities of strains  $\dot{\epsilon}_{ij}$  are linked with a vector of velocities  $\bar{U} = (u_x, u_y, u_z) = (u_1, u_2, u_3)$  by the following relation:

$$\dot{\epsilon}_{ij} = \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \quad (12.14)$$

where  $x_i$ ,  $x_j$  – *cartesian coordinates*. In addition, the following functions:

$$\tau_{ij}(u) = \varphi(l_2(u)) \dot{\epsilon}_{ij}(u) \quad (12.15)$$

are components of *deviator of stresses*. The *viscosity of a fluid* is represented by several expressions of the following form:

$$\varphi(l_2) = 2 \left( \frac{\sigma_0}{l_2} + k l_2^{n-1} \right), \quad (12.16)$$

$$\text{or } \varphi(l_2) = \left( \sum_{i=1}^n a_i l_2^{i-1} \right)^{-1} \quad (12.17)$$

In particular cases from the relations (12.16) - (12.17) can be obtained the generalized rheological

equations of Shvedov-Bingam ( $\mathbf{n}=1$ ), Newton ( $\boldsymbol{\sigma}_0=0$ ,  $\mathbf{k}=\boldsymbol{\mu}$ ) and Osval'd de Vil' ( $\boldsymbol{\sigma}_0=0$ ). The components of a stress tensor satisfy to the equations system of a motion of the following form:

$$\rho \frac{\partial u_i}{\partial t} + \sum_{j=1}^3 \left( \rho u_j \frac{\partial u_i}{\partial x_j} - \frac{\partial \sigma_{ij}(\mathbf{u})}{\partial x_j} \right) = F_i, \quad (i = 1, 2, 3) \quad (12.18)$$

where:  $\rho$  - density of a fluid;  $t$  - time;  $F_i$  - components of volumetric forces. By taking into account (12.12) and (12.14), the equation of motion (12.18) accepts the following form:

$$\rho \frac{\partial u_i}{\partial t} + \sum_{j=1}^3 \left\{ \rho u_j \frac{\partial u_i}{\partial x_j} + \frac{\partial \rho}{\partial x_j} \delta_{ij} - \frac{\partial}{\partial x_j} \left[ \varphi(l_2(\mathbf{u})) \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] \right\} = F_i, \quad (i = 1, 2, 3) \quad (12.19)$$

Equation of motion (12.19) further is considered. By discarding the inertial terms, we obtain the Stokes equation for a nonlinear viscous fluid as follows:

$$\frac{\partial \rho}{\partial x_i} - \sum_{j=1}^3 \frac{\partial}{\partial x_j} \left[ \varphi(l_2(\mathbf{u})) \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \right] = F_i, \quad (i = 1, 2, 3)$$

The usual guess in the classical hydromechanics is, that the fluid *adheres* to any solid surface. These conditions are fulfilled and for *non-Newtonian fluids*. However under some conditions the velocity of a *non-Newtonian* fluid on a solid surface can be different from velocity of the surface, i.e. takes place a *slippage* of a fluid on a surface. The *boundary conditions* on a surface are defined by the following relations:

$$\mathbf{u}|_{\Gamma} = \mathbf{u}_0 - \text{allocation of velocities of a fluid on boundary}; \quad (12.20)$$

on boundary  $\Gamma_1$  a fluid slips on a surface [73], namely:

$$\varphi(l_2(\mathbf{u})) \left[ \sum_{j=1}^3 \dot{\epsilon}_{ij}(\mathbf{u}) \mathbf{n}_j - \sum_{i=1}^3 \sum_{j=1}^3 \dot{\epsilon}_{ij}(\mathbf{u}) \mathbf{n}_i \mathbf{n}_j \mathbf{n}_k \right] = -\alpha(\mathbf{u}_\tau) \mathbf{u}_{\tau k}, \quad (k = 1, 2, 3), \quad (12.21)$$

where  $\mathbf{u}_{\tau k}$  - components of velocity, tangential to a surface  $\Gamma_1$  and  $\mathbf{n}_i$  - components of a unit vector, normal to the surface  $\Gamma_1$ .

The quantity  $\alpha$  is known in the literature as the slippage function of a fluid. The slippage function accepts positive value, while a negative sign in the formula (12.21) indicates, that the frictional force is directed into the side, opposite to motion. The *boundary conditions* (12.21) correspond to the condition, what the velocity of slide of a fluid in a point on a surface has a direction, opposite to projections of external force to a plane, tangential to the surface  $\Gamma_1$  in the given point; over value the velocity equals  $|\alpha(\mathbf{u}_\tau) \mathbf{u}_\tau|$ . The partial case of a rectilinear motion in the cylindrical channel of arbitrary cross-section is considered. In addition, the velocity  $\bar{\mathbf{U}}$  is directed along axis  $\mathbf{X}_3 = \mathbf{Z}$ , i.e.:

$$\mathbf{u}_x = \mathbf{u}_1 = 0, \quad \mathbf{u}_y = \mathbf{u}_2 = 0, \quad \mathbf{u}_z = \mathbf{u}_3 = \mathbf{u}(x, y)$$

Therefore, the components of a tensor of velocity of strain are defined as:

$$\dot{\epsilon}_{31} = \dot{\epsilon}_{zx} = \frac{\partial u}{\partial x}; \quad \dot{\epsilon}_{32} = \dot{\epsilon}_{zy} = \frac{\partial u}{\partial y}; \quad \dot{\epsilon}_{11} = \dot{\epsilon}_{22} = \dot{\epsilon}_{33} = \dot{\epsilon}_{12} = 0 \quad (12.22)$$

Then the second invariant of a tensor of velocities of strain becomes:

$$I_2 = 2 \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2 \right] \quad (12.23)$$

and the equation of motion of a non-Newtonian fluid is defined as follows:

$$\frac{\partial}{\partial x} \left( \varphi(I_2(u)) \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( \varphi(I_2(u)) \frac{\partial u}{\partial y} \right) = q \quad (12.24)$$

where  $q = F_z - \frac{\partial p}{\partial z}$ , and pressure drop  $\frac{\partial p}{\partial z}$  - constant quantity ( $\frac{\partial p}{\partial z} = \text{const}$ ).

The boundary conditions on a surface are defined by the following relations:

$$u|_{\Gamma} = u_0, \quad \text{where } u_0 - \text{velocity of a surface (if exist no slippage)} \quad (12.25)$$

$$\varphi(I_2(u)) \frac{\partial u}{\partial n} + \alpha(u) u = 0, \quad \text{motion with a slippage at the surface } \Gamma \quad (12.26)$$

For equation of motion of non-Newtonian fluid (12.24) and boundary conditions (12.26) it is possible to write the next determinative functional:

$$I(u) = \int_A \left\{ \varphi(I_2(u)) \left[ \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2 \right] - qu \right\} dA + \int_{\Gamma} \alpha(u) u^2 d\Gamma \quad (12.27)$$

The equation of motion (12.24) is solved by means of the FEM, for that the nine-nodal quadratic isoparametric finite element is used (fig. 12.2). The velocity in such finite element is approximated by the next expression:

$$u(r, s) = \sum_{i=1}^9 N_i(r, s) u_i = [N(r, s)] \{u\} \quad (12.28)$$

where:  $N_i$  - functions of the forms being defined from the expressions (12.6). By substituting (12.28) into the functional (12.27) and by equalling zero its variation, we obtain an equation of motion of a non-Newtonian fluid for a finite element  $e$  as follows:

$$[k^{(e)}(u)] \{u\} = \{f^{(e)}\}, \quad (12.29)$$

$$\text{where } [k^{(e)}(u)] = \int_{A^{(e)}} \varphi(I_2(u)) \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA + \int_{\Gamma^{(e)}} \alpha(u) [N]^T [N] d\Gamma; \quad (12.30)$$

$$\{f^{(e)}\} = \int_{A^{(e)}} q [N]^T dA.$$

By taking into account the relations (12.29) and (12.30), the global equations system of the following view is formed:

$$[\mathbf{K}(\mathbf{U})]\{\mathbf{U}\} = \{\mathbf{F}\} \quad (12.31)$$

The nonlinear system of the algebraic equations (12.31) is solved by the Newton method, namely:

$$[\mathbf{J}(\mathbf{U})]_i \{\Delta \mathbf{U}\}_i = -\{\mathbf{R}\}_i, \quad (12.32)$$

where  $[\mathbf{J}(\mathbf{U})]_i$  – Jacobi matrix;

$$[\mathbf{J}(\mathbf{U})]_i = \frac{\partial}{\partial \{\mathbf{U}\}^T} ([\mathbf{K}(\mathbf{U})]\{\mathbf{U}\} - \{\mathbf{F}\})_i; \quad (12.33)$$

$$\{\mathbf{R}\}_i = [\mathbf{K}(\mathbf{U})]_i \{\mathbf{U}\}_i - \{\mathbf{F}\}_i \quad (12.34)$$

In addition, the *vector of velocity* on each *i*-th iteration is defined as follows:

$$\{\mathbf{U}\}_i = \{\mathbf{U}\}_{i-1} + \{\Delta \mathbf{U}\}_i \quad (12.35)$$

At smooth change of velocity during iterations the most simple condition of reaching of the solution is used, namely:

$$\max |\{\Delta \mathbf{U}\}_i| < \varepsilon \quad (12.36)$$

The *volumetric consumption*  $Q$  and a *mean velocity*  $u_{av}$  in cross-section of a channel are defined by the following relations:

$$Q = \sum_{e=1}^{NE} \int_{A^{(e)}} u^{(e)}(x, y) dA; \quad (12.37)$$

$$u_{av} = \frac{Q}{\sum_{e=1}^{NE} \int_{A^{(e)}} dA} \quad (12.38)$$

where  $NE$  – total number of the used finite elements.

### 12.2.2. Input data for the solution of the problem

For the solution of *equation of motion* of a *non-Newtonian* fluid in a cylindrical channel of arbitrary cross-section, the *NonNiuton* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable  $\mathbf{F}$ . The data in the file are placed in the strict order. In the *first* line the next parameters are coded: the number of finite elements (*parameter nelem*), number of nodes (*npoim*), the number of sides of finite elements on a surface, where slippage of a fluid (*ngrad*) is preset, number of intervals of function  $I_2$  (*nab*) and the number of coefficients of a regression describing the dependence of function of viscosity from the second invariant of a tensor of velocities of the strain (*nvar*), i.e.:

$$\varphi(I_2) = a_0 + \sum_{i=1}^{(nvar-1)/2} a_i I_2^{b_i} \quad (12.39)$$

In the *second* line the next parameters are coded: a print code of intermediate results (*kprint*), the number of iterations (*niteration*), precision of the solution (*tol*), pressure drop onto unity of length of a channel (*dpress*) and density of a fluid (*tankis*). If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. In the *third* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). If parameter *ksolve*=0, then for the solution, the *solve* function of the Maple-language is used; otherwise, the equations system by the method of *conjugate gradients* is solved. In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where *nnode* – number of nodes of the finite element, i.e. *nnode*=9} are coded.

After of array **Mtop** the elements of **Lgrad**(*ngrad*, 4) array are coded line by line. Into each line of the file are recorded the line number of the array **Lgrad**, the number of a finite element and also three numbers of nodes located on a side of finite element, laying on a surface, where happens a slippage of a fluid. Behind of array **Lgrad** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, into the *second* column of the **Coord** array the *y*-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded.

Behind of array **Coord** the elements of **Grad**(*ngrad*, 3) array are recorded line by line. Into each line of the datafile a line number of **Grad** array and the values of function of a slippage in nodes, which are on a surface, are coded. The lines of **Grad** array should strictly correspond to the lines of **Lgrad** array. If on a canal surface of slippage of a fluid not exists, then parameter *ngrad*=1 and for the first element of the array **Lgrad** the value -1 is set, i.e. **Lgrad**[1, 1]=-1, while for the elements of the array **Grad** any values are set (for example, **Grad**[1, 1]= 0, **Grad**[1, 2]=0, **Grad**[1, 3]=0).

Behind of array **Grad** into the datafile the elements of **Avar**(*nab*, *nvar*) array are recorded line by line. In this array the values of coefficients **a**<sub>0</sub>, **a**<sub>1</sub>, **b**<sub>1</sub>, ..., **a**<sub>*nvar* - 1</sub>, **b**<sub>*nvar*</sub> are describe function of viscosity  $\varphi(l_2)$  according to the expression (12.39). Into each line of the datafile a line number of **Avar** array and the values of coefficients **a**<sub>*i*</sub>, **b**<sub>*i*</sub> are recorded. Behind of array **Avar** into the datafile the elements of **Abond**(*nab*, 2) array are recorded line by line. In this array the values of minimum and maximum of **l**<sub>2</sub> are described. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, ngrad, nab, nvar, kprint, niteration, tol, dpress, tankis, ksolve, niter, toler*  
 Text line \*  
 Arrays **Mtop**(*nelem*, *nnode*)  
 Text line \*  
 Array **Lgrad**(*ngrad*, 4)  
 Text line \*  
 Array **Coord**(*npoin*, *ndime*)  
 Text line \*  
 Array **Grad**(*ngrad*, 3)  
 Text line \*  
 Array **Avar**(*nab*, *nvar*)  
 Text line \*  
 Array **Abond**(*nab*, 2)

### 12.2.3. Brief description of the NonNiuton program solving the problem

The *NonNiuton* program was programmed on the *Maple*-language; it consists of the basic program and 36 procedures. All procedures can be divided into three groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates the values of velocity in nodes of finite elements, medial velocity of a motion and volumetric consumption in a quarter of the cross-section of the channel. The calculation results are output on the monitor and are recorded into files, for that to variable *file\_rez1* must be ascribed qualifiers of the target files. In this file the values of velocity in nodes of finite elements, medial velocity of a motion and volumetric consumption in a quarter of the cross-section of the channel are recorded.

### 12.2.4. An example of use of the Maple-program NonNiuton

The motion of a warmed-up steel with 0.06% of carbon is considered at temperature  $T=927$  K in the cylindrical channel of rectangular cross-section of the size  $0.6 \cdot 10^{-2} \times 0.4 \cdot 10^{-2}$  m. The dependence of function of viscosity, obtained according to the data [73], accepts the following form:

$\varphi(l_2) = a_0 + \sum_{i=1} a_i \cdot l_2^{b_i}$ , where:  $[l_2] = s^{-2}$ ,  $[\varphi] = N \cdot s / m^2$ . As the rectangular cross-section of the

channel has two planes of symmetry, then the *quarter* of cross section is considered only, i.e.  $0.3 \cdot 10^{-2} \times 0.2 \cdot 10^{-2}$  m. The calculated scheme of cross-section of the cylinder, in which the motion of warmed-up steel is investigated, is represented on the *fig. 12.6*.

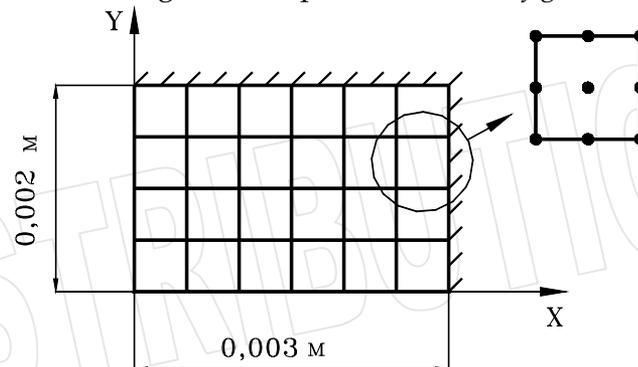


Fig. 12.6. The calculated scheme of warmed-up steel in cylindrical channel of rectangular cross-section

Input data for the test example: *nelem*=24, *npoin*=117, *nvar*=3, *nbond*=21, *ngrad*=1, *kprint*=0, *ksolve* = 0, *niter* =50, *toler*=106(-6), *tol*=10<sup>^</sup>(-6), *niteration*=10. In addition, the following settings are supposed: a pressure drop onto unity of length of the channel *dpress*=-7x10<sup>^</sup>10 N/m<sup>^</sup>3, density of steel *tankis*=0 (the volumetric forces are ignored), velocity of a fluid on the channel surface  $u_0 = 0$ ;  
 $a_1 = 1,4406 \cdot 10^{-9}$ ,  $a_2 = 1,6857 \cdot 10^{-8}$ ,  $a_3 = -1,3755 \cdot 10^{-8}$ .

#### Results of calculation over the test example:

The values of velocity (*veloc*) in nodes (*node*) of finite elements are determined as:

node=1	veloc=2.838330e-04	node=6	veloc=2.527976e-04
node=2	veloc=2.826298e-04	node=7	veloc=2.384733e-04
node=3	veloc=2.790006e-04	node=8	veloc=2.209288e-04
node=4	veloc=2.728904e-04	node=9	veloc=1.996855e-04
node=5	veloc=2.642028e-04	node=10	veloc=1.738145e-04

node=11	veloc=1.412735e-04	node=54	veloc=2.297814e-04
node=12	veloc=9.273785e-05	node=55	veloc=2.245714e-04
node=13	veloc=0e-01	node=56	veloc=2.084091e-04
node=14	veloc=2.818784e-04	node=57	veloc=1.802292e-04
node=15	veloc=2.770342e-04	node=58	veloc=1.393253e-04
node=16	veloc=2.621933e-04	node=59	veloc=8.441416e-05
node=17	veloc=2.363770e-04	node=60	veloc=0e-01
node=18	veloc=1.974569e-04	node=61	veloc=1.995164e-04
node=19	veloc=1.388880e-04	node=62	veloc=1.981947e-04
node=20	veloc=0e-01	node=63	veloc=1.941775e-04
node=21	veloc=2.759323e-04	node=64	veloc=1.873291e-04
node=22	veloc=2.747176e-04	node=65	veloc=1.774988e-04
node=23	veloc=2.710513e-04	node=66	veloc=1.645896e-04
node=24	veloc=2.648724e-04	node=67	veloc=1.486632e-04
node=25	veloc=2.560767e-04	node=68	veloc=1.299540e-04
node=26	veloc=2.445176e-04	node=69	veloc=1.089144e-04
node=27	veloc=2.299920e-04	node=70	veloc=8.595387e-05
node=28	veloc=2.122034e-04	node=71	veloc=6.125417e-05
node=29	veloc=1.906835e-04	node=72	veloc=3.380427e-05
node=30	veloc=1.645189e-04	node=73	veloc=0e-01
node=31	veloc=1.317796e-04	node=74	veloc=1.380701e-04
node=32	veloc=8.473923e-05	node=75	veloc=1.341596e-04
node=33	veloc=0e-01	node=76	veloc=1.212442e-04
node=34	veloc=2.657387e-04	node=77	veloc=9.789697e-05
node=35	veloc=2.607926e-04	node=78	veloc=6.657815e-05
node=36	veloc=2.455780e-04	node=79	veloc=3.413601e-05
node=37	veloc=2.190212e-04	node=80	veloc=0e-01
node=38	veloc=1.791080e-04	node=81	veloc=0e-01
node=39	veloc=1.200949e-04	node=82	veloc=0e-01
node=40	veloc=0e-01	node=83	veloc=0e-01
node=41	veloc=2.507700e-04	node=84	veloc=0e-01
node=42	veloc=2.495153e-04	node=85	veloc=0e-01
node=43	veloc=2.457198e-04	node=86	veloc=0e-01
node=44	veloc=2.392988e-04	node=87	veloc=0e-01
node=45	veloc=2.301237e-04	node=88	veloc=0e-01
node=46	veloc=2.180356e-04	node=89	veloc=0e-01
node=47	veloc=2.028585e-04	node=90	veloc=0e-01
node=48	veloc=1.843708e-04	node=91	veloc=0e-01
node=49	veloc=1.622700e-04	node=92	veloc=0e-01
node=50	veloc=1.359420e-04	node=93	veloc=0e-01
node=51	veloc=1.041682e-04	node=94	veloc=2.806724e-04
node=52	veloc=6.282364e-05	node=95	veloc=2.709072e-04
node=53	veloc=0e-01	node=96	veloc=2.507507e-04

node=97	veloc=2.187716e-04	node=108	veloc=1.958777e-04
node=98	veloc=1.715033e-04	node=109	veloc=1.613878e-04
node=99	veloc=9.070861e-05	node=110	veloc=1.138463e-04
node=100	veloc=2.645086e-04	node=111	veloc=4.871725e-05
node=101	veloc=2.545205e-04	node=112	veloc=1.371204e-04
node=102	veloc=2.338112e-04	node=113	veloc=1.289579e-04
node=103	veloc=2.009290e-04	node=114	veloc=1.108521e-04
node=104	veloc=1.527255e-04	node=115	veloc=8.283526e-05
node=105	veloc=7.518445e-05	node=116	veloc=5.013938e-05
node=106	veloc=2.284890e-04	node=117	veloc=1.825232e-05
node=107	veloc=2.179250e-04		

Medial velocity and a volumetric consumption in a quarter of cross-section of the channel equal  $u_{av} = 0,1615 \cdot 10^{-3} \text{ m/s}$  and  $Q = 9,6903 \cdot 10^{-10} \text{ m}^3/\text{s}$  accordingly. The graph of allocation of velocity of a motion of warmed-up steel in a quarter of rectangular cross-section of the channel is represented on the fig. 12.7.

Velocity  $u(x,y)$  Distribution:

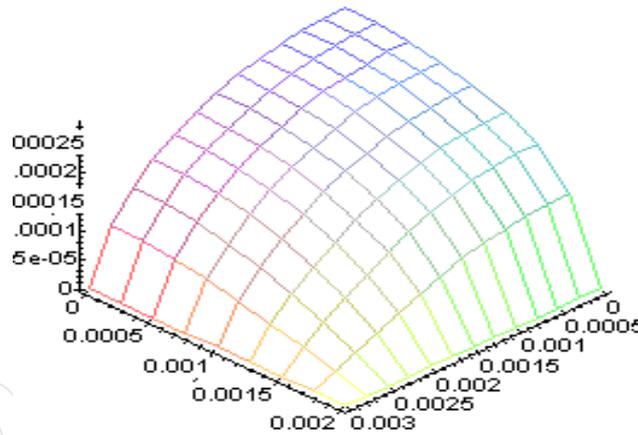
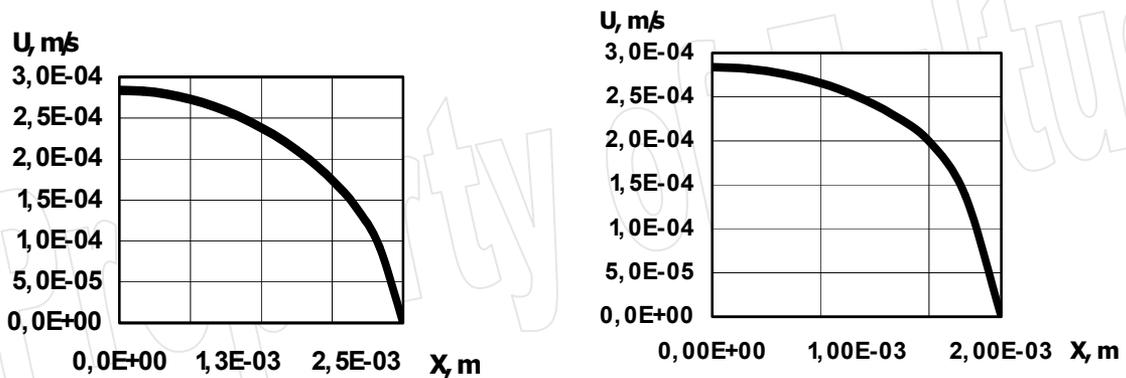


Fig. 12.7. The graph of allocation of velocity of a motion of warmed-up steel in a quarter of rectangular cross-section of the channel

The graphs of allocation of velocities of a motion of warmed-up steel along axes X (a) and Y (b) are represented on the fig. 12.8.



(a) along axes X

(b) along axes Y

Fig. 12.8. The graphs of allocation of velocities of motion of warmed-up steel

The *NonNewton* program is intended for the solution of an equation of motion of a *non-Newtonian* fluid in the cylindrical channel of arbitrary cross-section. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are presented in the PROBLEMS directory of archive attached to the present book in datafiles Mb9\_2\_new.mws, Mb9\_2.dat and Mb9\_2\_1.rez accordingly.

### 12.3. A model of convective heat exchange in a fluid

The *nonisothermal* motion of a fluid considerably differs from *isothermal*, because in this case at change of temperature the physical properties of a fluid vary, in this connection in the fluid arise qualitatively new phenomena. The concept of *convective heat exchange* embraces process of heat exchange at a motion of a fluid and gas, when the transport of heat is carried out simultaneously both by *convection*, and *thermal conduction*. Under *convection of heat* understand a carrying over of heat at moving of particles of a fluid or gas in space from area with one temperature into area with another temperature. The convection can be only in a moving medium, where the transport of heat is inseparably linked with transport of a medium. Such physical processes are rather widely used in technique, therefore solution of similar problems represents major practical interest.

#### 12.3.1. The calculated equations of nonisothermal convective heat exchange

The *nonisothermal motion* of a fluid explored on the basis of the non-stationary equations of Navier-Stokes together with the equations of transport of heat, in the Bussinesk's approximation accepts the following view [72]:

$$\begin{aligned} \frac{\partial u_x}{\partial t} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{\text{Re}} \left( \Delta u_x - \frac{\gamma}{x^2} u_x \right) &= 0; \\ \frac{\partial u_y}{\partial t} + u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{\text{Re}} \Delta u_y - \frac{\text{Gr}}{\text{Re}^2} T &= 0; \\ \frac{\partial u_x}{\partial x} + \frac{\gamma}{x} u_x + \frac{\partial u_y}{\partial y} &= 0; \quad \frac{\partial T}{\partial t} + u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y} - \frac{1}{\text{RePr}} \Delta T &= 0 \end{aligned} \quad (12.40)$$

where:  $u_x, u_y$  - components of velocity of a fluid in direction of axes X and Y;  $p$  - pressure;  $T$  - temperature;  $\text{Re}$  - the Reynolds number ( $\text{Re} = \frac{vL}{\nu}$ ,  $v$  - scale of velocity);  $L$  - characteristic of the

size;  $\nu$  - kinematic viscosity);  $\text{Gr}$  - the *Grashof* number ( $\text{Gr} = \frac{g\beta_T L^3 \Delta T}{\nu^2}$ ,  $g$  - acceleration of gravity);

$\beta_T$  - coefficient of thermal change of density ( $\beta_T = -\frac{1}{\rho} \frac{\partial \rho}{\partial T}$ );  $\Delta T$  - difference of temperatures;  $\text{Pr}$  -

Prandtl number ( $\text{Pr} = \frac{v}{a}$ ,  $a = \frac{\lambda}{\rho c_p}$  - thermal diffusivity);  $\lambda$  - thermal conductivity;  $c_p$  - coefficient

of a specific heat capacity at constant pressure;  $\gamma = 0$ , using cartesian coordinates, and  $\gamma = 1$ , using cylindrical axials at an axial-symmetric motion ( $x=r, y=z$ ). The Laplace operator has the following well known form:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\gamma}{x} \frac{\partial}{\partial x} + \frac{\partial^2}{\partial z^2} \quad (12.41)$$

The equations system (12.40) is solved by a method of correction of velocity. Using the given method, the new variables of velocity  $\underline{u}_{x,n}$  and  $\underline{u}_{y,n}$  are introduced, where index  $n$  indicates the

number of integration step. Then, the *first* and *second* equations from the system (12.40), by discarding the terms  $\frac{\partial p}{\partial x}$  and  $\frac{\partial p}{\partial y}$ , can be written as follows:

$$\begin{aligned} \frac{u_{x,n} - u_{x,n}}{\tau} + u_{x,n} \frac{\partial u_{x,n}}{\partial x} + u_{y,n} \frac{\partial u_{x,n}}{\partial y} - \frac{1}{\text{Re}} \left( \Delta u_{x,n} - \frac{\gamma}{x^2} u_{x,n} \right) &= 0; \\ \frac{u_{y,n} - u_{y,n}}{\tau} + u_{x,n} \frac{\partial u_{y,n}}{\partial x} + u_{y,n} \frac{\partial u_{y,n}}{\partial y} - \frac{1}{\text{Re}} \Delta u_{y,n} - \frac{\text{Gr}}{\text{Re}^2} T_n &= 0 \end{aligned} \quad (12.42)$$

where:  $\tau$  - integration step in a time. Then the derivatives of velocities  $u_x$  and  $u_y$  in a time are calculated as follows:

$$\frac{\partial u_x}{\partial t} = \frac{u_{x,n+1} - u_{x,n}}{\tau}; \quad \frac{\partial u_y}{\partial t} = \frac{u_{y,n+1} - u_{y,n}}{\tau} \quad (12.43)$$

By substituting derivatives of velocities in a time (12.43) into the first two equations (12.40) and by eliminating variables  $u_{x,n}$  and  $u_{y,n}$ , we obtain the equations for pressure of the following form:

$$\frac{\partial u_{x,n+1}}{\partial t} - \frac{\partial u_{x,n}}{\partial t} = - \frac{\partial p_{n+1}}{\partial x}; \quad \frac{\partial u_{y,n+1}}{\partial t} - \frac{\partial u_{y,n}}{\partial t} = - \frac{\partial p_{n+1}}{\partial y} \quad (12.44)$$

By using now continuity equation {the third equation in (12.40)} together with the equations (12.41), we obtain the Poisson equation for variable of pressure:

$$\frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma \frac{\partial p_{n+1}}{\partial x} \right) + \frac{\partial^2 p_{n+1}}{\partial y^2} = \frac{1}{\tau} \left[ \frac{1}{x^\gamma} \frac{\partial (x^\gamma u_{x,n})}{\partial x} + \frac{\partial x u_{y,n}}{\partial y} \right] \quad (12.45)$$

By expressing derivative of temperature in a time, we obtain the next relation:

$$\frac{\partial T}{\partial t} = \frac{T_{n+1} - T_n}{\tau} \quad (12.46)$$

while a heat conduction equation {the fourth equation in (12.40)} is reduced to the following form:

$$T_{n+1} = T_n - \tau \left( u_{x,n+1} \frac{\partial T_n}{\partial x} + u_{y,n+1} \frac{\partial T_n}{\partial y} \right) - \frac{\tau}{\text{RePr}} \left[ \frac{1}{x^\gamma} \frac{\partial}{\partial x} \left( x^\gamma \frac{\partial T_n}{\partial x} \right) + \frac{\partial^2 T_n}{\partial y^2} \right] \quad (12.47)$$

Thus, for definition of values of variables ( $u_x$ ,  $u_y$ ,  $p$ ,  $T$ ) on an integration step  $n+1$  it is necessary to solve the equations system (12.42), (12.44), (12.45), and (12.47). Such system of the differential equations is solved by means of the FEM, for that the *four-nodal linear isoparametric* finite element is used; the *velocity*, the *pressure* and *temperature* in the element are approximated by the following relation:

$$\varphi(r, s) = \sum_{i=1}^4 N_i(r, s) \varphi_i = [N(r, s)]\{\varphi\}, \text{ where: } \varphi \in (u_x, u_y, p, T). \quad (12.48)$$

By applying the Galerkin's procedure, the global equations system of convective heat exchange accepts the following common view, namely:

$$[M]\{u_{x,n}\} = [M]\{u_{x,n}\} + \tau \{F_{ux}\} - ([A] + [K_{ux}])\{u_{x,n}\};$$

$$\begin{aligned} [M]\{U_{y,n}\} &= [M]\{U_{y,n}\} + \tau\{F_{uy}\} + \{F_{uyt}\} - ([A] + [K_{uy}])\{U_{y,n}\}; [K_p]\{P_{n+1}\} = \{F_{p_1}\} - \{F_{p_2}\}; \\ [M]\{U_{x,n+1}\} &= [M]\{U_{x,n}\} - \tau\{R_{ux}\}; [M]\{U_{y,n+1}\} = [M]\{U_{y,n}\} - \tau\{R_{uy}\}; \\ [M]\{T_{n+1}\} &= [M]\{T_n\} + \tau\{F_T\} + ([A] + [K_T])\{T_n\}, \end{aligned} \quad (12.49)$$

where  $[M] = \sum_{e=1}^{NE} [m^{(e)}]$ ,  $[m^{(e)}] = \int_{A^{(e)}} [N]^T [N] dA$ ;  $[K_p] = \sum_{e=1}^{NE} [k_p^{(e)}]$ ,

$$[k_p^{(e)}] = \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA; \{a\} = \sum_{e=1}^{NE} \{a^{(e)}\},$$

$$\{a^{(e)}\} = \int_{A^{(e)}} [N]^T \left( u_{x,n} \left[ \frac{\partial N}{\partial x} \right] + u_{y,n} \left[ \frac{\partial N}{\partial y} \right] \right) dA; [K_{ux}] = \sum_{e=1}^{NE} [k_{ux}^{(e)}],$$

$$[k_{ux}^{(e)}] = \frac{1}{Re} \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] - \frac{\gamma}{x^2} [N]^T [N] \right) dA; \{F_{ux}\} = \sum_{e=1}^{NE} \{f_{ux}^{(e)}\},$$

$$\{f_{ux}^{(e)}\} = \frac{1}{Re} \int_{\Gamma^{(e)}} [N]^T x \gamma \left( \frac{\partial u_{x,n}}{\partial x} l_x + \frac{\partial u_{x,n}}{\partial y} l_y \right) d\Gamma; [K_{uy}] = \sum_{e=1}^{NE} [k_{uy}^{(e)}],$$

$$[k_{uy}^{(e)}] = \frac{1}{Re} \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA; \{F_{uy}\} = \sum_{e=1}^{NE} \{f_{uy}^{(e)}\},$$

$$\{f_{uy}^{(e)}\} = \frac{1}{Re} \int_{\Gamma^{(e)}} [N]^T x \gamma \left( \frac{\partial u_{y,n}}{\partial x} l_x + \frac{\partial u_{y,n}}{\partial y} l_y \right) d\Gamma; \{F_{uyt}\} = \sum_{e=1}^{NE} \{f_{uyt}^{(e)}\},$$

$$\{f_{uyt}^{(e)}\} = \frac{Gr}{Re^2} \int_{A^{(e)}} [N]^T [N] \{T_n\} dA; \{F_{p_1}\} = \sum_{e=1}^{NE} \{f_{p_1}^{(e)}\}, \quad (12.50)$$

$$\{f_{p_1}^{(e)}\} = \int_{\Gamma^{(e)}} [N]^T x \gamma \left( \frac{\partial p_n}{\partial x} l_x + \frac{\partial p_n}{\partial y} l_y \right) d\Gamma; \{F_{p_2}\} = \sum_{e=1}^{NE} \{f_{p_2}^{(e)}\},$$

$$\{f_{p_2}^{(e)}\} = \frac{1}{\tau} \int_{A^{(e)}} [N]^T \left( \frac{\partial u_{x,n}}{\partial x} + \frac{\partial u_{y,n}}{\partial y} + u_{x,n} \right) dA; \{R_{ux}\} = \sum_{e=1}^{NE} \{r_{ux}^{(e)}\}, \{r_{ux}^{(e)}\} = \int_{A^{(e)}} [N]^T \frac{\partial p_{n+1}}{\partial x} dA;$$

$$\{R_{uy}\} = \sum_{e=1}^{NE} \{r_{uy}^{(e)}\}, \{r_{uy}^{(e)}\} = \int_{A^{(e)}} [N]^T \frac{\partial p_{n+1}}{\partial y} dA; [K_T] = \sum_{e=1}^{NE} [k_T^{(e)}],$$

$$\{k_T^{(e)}\} = \frac{1}{\text{RePr}} \int_{A^{(e)}} [N]^T \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA; \{F_T\} = \sum_{e=1}^{NE} \{f_T^{(e)}\},$$

$$\{f_T^{(e)}\} = \frac{1}{\text{RePr}} \int_{\Gamma^{(e)}} [N]^T \left( \frac{\partial T_n}{\partial x} I_x + \frac{\partial T_n}{\partial y} I_y \right) d\Gamma;$$

$I_x, I_y$  – direction cosines;  $NE$  – total number of the finite elements. The equations system (12.49) is solved with the following *boundary* and *initial* conditions:

$$u_x|_{\Gamma} = u_{x_r}, \quad u_y|_{\Gamma} = u_{y_r}; \quad p|_{\Gamma} = p_r; \quad T|_{\Gamma} = T_r; \quad \frac{\partial u_x}{\partial x}|_{\Gamma}; \quad \frac{\partial u_x}{\partial y}|_{\Gamma}; \quad \frac{\partial u_y}{\partial x}|_{\Gamma};$$

$$\frac{\partial u_y}{\partial y}|_{\Gamma}; \quad \frac{\partial p}{\partial x}|_{\Gamma}; \quad \frac{\partial p}{\partial y}|_{\Gamma}; \quad \frac{\partial T}{\partial x}|_{\Gamma}; \quad \frac{\partial T}{\partial y}|_{\Gamma}; \quad u_x|_{t=0} = u_{x_0}(x, y), \quad (12.51)$$

$$u_y|_{t=0} = u_{y_0}(x, y); \quad p|_{t=0} = 0; \quad T|_{t=0} = T_0(x, y)$$

By solving the given problem by a method of correction of velocity, it is not required to form the matrixes  $[M]$  and  $[K_p]$  on each integration step, but it is required to calculate only *right parts* of the equations system. However, for reaching a stability of the solution it is necessary to use a rather small integration step.

### 12.3.2. Input data for the solution of the problem

For the solution of the equations system describing a convective heat exchange in some plane, the *Heat\_flow* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order.

In the *first* line the next parameters are coded: the number of finite elements (parameter *nelem*), number of nodes (*npoin*), the number of Dirichlet's boundary conditions (*nbond*) for *velocity* in direction of axes  $X$  and  $Y$  (*nbundu, nbondv*), for *pressure* (*nbondp*) and *temperature* (*nbondt*). Into the *second* line the number of Neumann's boundary conditions for *velocity* in direction of axes  $X$  and  $Y$  (*ngradu, ngradv*), for *pressure* (*ngradp*) and *temperature* (*ngradt*) are recorded.

In the *third* line the next parameters are coded: a print code of intermediate results (*kprint*), a type of a problem (*ngama*), the Reynolds number (*reinold*), the Prandtl number (*prandl*) and the Grashof number (*grashof*) are indicated. If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0, otherwise – in *cylindrical* frames is solved.

In the *fourth* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). If parameter *ksolve*=0, then for the solution, the *'solve'* function of the *Maple*-language is used, otherwise the equations system by the method of *conjugate gradients* is being solved. In the *fifth* line, the number of integration steps (*ntime*) and also size of an integration step (*dtime*) are coded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element (*array Mtop(nelem, nnode)*, where *nnode* – number of nodes of a finite element, i.e. *nnode* =

4) are coded. After of array **Mtop** the elements of **Lbondu**(*nbondu*) array are coded line by line. Into each line of the file the line number and element of **Lbondu** array are recorded. Into the **Lbondu** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *velocity* in direction of axis **X**, are coded.

After of array **Lbondu** the elements of **Lbondv**(*nbondv*) array are coded line by line. Into each line of the file the line number and element of **Lbondv** array are recorded. Into the **Lbondv** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *velocity* in direction of axis **Y**, are coded. After of array **Lbondv** the elements of **Lbondp**(*nbondp*) array are coded line by line. Into each line of the file the line number and element of **Lbondp** array are recorded. Into the **Lbondp** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *pressure*, are coded.

After of array **Lbondp** the elements of **Lbondt**(*nbondt*) array are coded line by line. Into each line of the file the line number and element of **Lbondt** array are recorded. Into the **Lbondt** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *temperature*, are coded. After of array **Lbondt** the elements of **Lgradu**(*ngradu*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradu** array are recorded. In this array the next parameters are coded: the line number of **Lgradu** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *velocity* in direction of axis **X** are preset.

After of array **Lbondu** the elements of **Lgradv**(*ngradv*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradv** array are recorded. In this array the next parameters are coded: the line number of **Lgradv** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *velocity* in direction of axis **Y** are preset. After of array **Lbondv** the elements of **Lgradp**(*ngradp*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradp** array are recorded. In this array the next parameters are coded: the line number of **Lgradp** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *pressure* are given. After of array **Lbondp** the elements of **Lgradt**(*ngradt*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradt** array are recorded. In this array the next parameters are coded: the line number of **Lgradt** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where the Neumann boundary conditions for *temperature* are given.

Behind of array **Lgradt** into the data file the elements of **Coord**(*npoin*, *ndime*) array, where *ndime* – dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the **x**-coordinates, into the *second* column of the **Coord** array the **y**-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (**x**, **y**)-coordinates are recorded. Behind of array **Coord** the elements of **Bondu**(*nbondu*) array are recorded line by line. Into each line of the datafile a line number of **Bondu** array and a value of *velocity* in direction of axis **X** are recorded. The lines of **Bondu** array should strictly correspond to the lines of **Lbondu** array.

Behind of array **Bondu** the elements of **Bondv**(*nbondv*) array are recorded line by line. Into each line of the datafile a line number of **Bondv** array and a value of *velocity* in direction of axis **Y** are recorded. The lines of **Bondv** array should strictly correspond to the lines of **Lbondv** array. Behind of array **Bondv** the elements of **Bondp**(*nbondp*) array are recorded line by line. Into each line of the datafile a line number of **Bondp** array and a value of *pressure* are recorded. The lines of **Bondp** array should strictly correspond to the lines of **Lbondp** array. Behind of array **Bondp** the elements of **Bondt**(*nbondt*) array are recorded line by line. Into each line of the datafile a line number of

**Bondt** array and a value of temperature are recorded. The lines of **Bondt** array should strictly correspond to the lines of **Lbondt** array.

Behind of array **Bondt** the elements of **Gradu(ngradu, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradu** array and also values of derivatives  $\frac{\partial u_x}{\partial x}$  and  $\frac{\partial u_x}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradu** array should strictly correspond to the lines of **Lgradu** array. Behind of array **Gradu** the elements of **Gradv(ngradv, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradv** array and also values of derivatives  $\frac{\partial u_y}{\partial x}$  and  $\frac{\partial u_y}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradv** array should strictly correspond to the lines of **Lgradv** array.

Behind of array **Gradv** the elements of **Gradp(ngradp, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradp** array and also values of derivatives  $\frac{\partial p}{\partial x}$  and  $\frac{\partial p}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradp** array should strictly correspond to the lines of **Lgradp** array. At last, after of array **Gradp** the elements of **Gradt(ngradt, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradt** array and also values of derivatives  $\frac{\partial T}{\partial x}$  and  $\frac{\partial T}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradt** array should strictly correspond to the lines of **Lgradt** array. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, nbondu, nbondv, nbondp, nbondt, ngradu, ngradv, ngradp, ngradt, kprint, ngama, reynold, prandl, grashof, ksolve, niter, toler*  
 Text line \*  
 Arrays **Mtop(nelem, nnode)**  
 Text line \*  
 Array **Lbondu(nbondu)**  
 Text line \*  
 Array **Lbondv(nbondv)**  
 Text line \*  
 Array **Lbondp(nbondp)**  
 Text line \*  
 Array **Lbondt(nbondt)**  
 Text line \*  
 Array **Lgradu(ngradu, 3)**  
 Text line \*  
 Array **Lgradv(ngradv, 3)**  
 Text line \*  
 Array **Lgradp(ngradp, 3)**  
 Text line \*  
 Array **Lgradt(ngradt, 3)**  
 Text line \*  
 Array **Coord(npoin, ndime)**

Text line \*  
 Array *Bondu*(*nbondu*)  
 Text line \*  
 Array *Bondv*(*nbondv*)  
 Text line \*  
 Array *Bondp*(*nbondp*)  
 Text line \*  
 Array *Bondt*(*nbondt*)  
 Text line \*  
 Array *Gradu*(*ngradu*, 2)  
 Text line \*  
 Array *Gradv*(*ngradv*, 2)  
 Text line \*  
 Array *Gradp*(*ngradp*, 2)  
 Text line \*  
 Array *Gradt*(*ngradt*, 2)

### 12.3.3. Brief description of the Heat\_flow program solving the problem

The *Heat\_flow* program was programmed on the *Maple*-language; it consists of the basic program and 52 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates the values of velocities in direction of axes *X* and *Y*, and also of pressure and temperature in nodes of the finite elements. The calculation results are output on the monitor and are recorded into files, for that to variable *file\_rez1* must be ascribed qualifier of the target file. In this file the values of velocities in direction of axes *X* and *Y*, and of pressure and temperature in nodes of the finite elements are recorded.

### 12.3.4. An example of use of the Maple-program Heat\_flow

A problem of a thermal convection in a closed square area of size (1.0x1.0) at isothermal side boundaries is considered. The boundary conditions for this problem are given as follows:

$$\begin{aligned}
 & \mathbf{x} = 0, \quad 0 \leq \mathbf{y} \leq 1, \quad u_x = u_y = 0, \quad p = 0, \quad T = 1; \quad \mathbf{x} = 1, \quad 0 \leq \mathbf{y} \leq 1; \quad u_x = u_y = 0, \quad T = 0; \quad \mathbf{y} = 0 \text{ and } \mathbf{y} = 1, \\
 & \quad 0 \leq \mathbf{x} \leq 1, \quad u_x = u_y = 0, \quad \frac{\partial T}{\partial \mathbf{y}} = 0
 \end{aligned}$$

The calculated scheme of the closed square area is represented on the *fig. 12.9*.

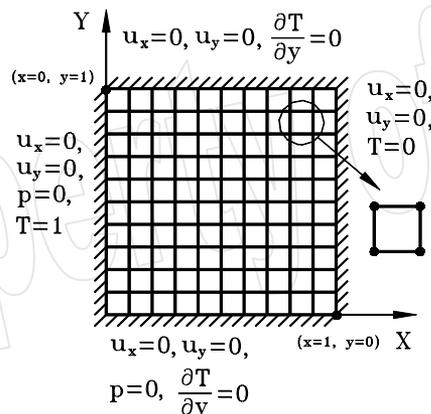


Fig. 12.9. The calculated scheme and boundary conditions in the square area

*Input data for the test example: nelem=100, npoin=121, nbondu=40, nbondv=40, nbondp=21, nbondt = 22, ngradu=1, ngradv=1, ngradp=1, ngradt=1, kprint=0, ngama=0, ksolve=1, niter=50, toler=10<sup>(-6)</sup>, ntime=9, dtime=5\*10<sup>(-4)</sup> s, Reynold number reinold=1, Prandtl number prandl=0.7, Grashof number grashof=10<sup>3</sup>.*

***Results of calculation over the test example:***

*The following table represents the values of velocities (ux, uy), pressure (press) and temperature (temp) in nodes (node) of finite elements, namely:*

node	Ux	uy	press	temp
1	0.0	0.0	0.0	1e+00
2	0.0	0.0	0.0	-1.638461e-01
3	0.0	0.0	0.0	2.686961e-02
4	0.0	0.0	0.0	-4.432926e-03
5	0.0	0.0	0.0	7.559319e-04
6	0.0	0.0	0.0	-1.481106e-04
7	0.0	0.0	0.0	4.063429e-05
8	0.0	0.0	0.0	-1.401222e-05
9	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	1e+00
11	1.013140e-03	5.629096e-04	-1.413853e+00	-1.638442e-01
12	-4.933456e-03	4.775714e-04	7.327017e-02	2.686990e-02
13	1.438302e-03	-2.312943e-04	2.670254e-02	-4.432753e-03
14	-5.638933e-04	-3.540205e-04	5.200196e-02	7.556664e-04
15	2.783736e-05	-4.488438e-04	9.253299e-02	-1.479035e-04
16	-1.379072e-05	-4.419744e-04	9.088716e-02	4.050747e-05
17	-6.652626e-06	-6.150403e-04	9.537795e-02	-1.395441e-05
18	0.0	0.0	9.425972e-02	0.0
19	0.0	0.0	0.0	1e+00
20	3.215258e-04	-4.910027e-03	2.215147e-01	-1.638460e-01
21	5.331771e-04	-9.676222e-05	-1.510565e-01	2.686946e-02
22	-1.070505e-0	-3.352683e-04	1.047730e-01	-4.433032e-03
23	8.802836e-05	-4.176571e-04	1.537977e-01	7.560026e-04
24	-1.244935e-04	-3.468258e-04	1.748362e-01	-1.481414e-04
25	-1.710248e-05	-3.387422e-04	1.892892e-01	4.064552e-05
26	-1.875573e-06	-4.471645e-04	1.910140e-01	-1.401547e-05
27	0.0	0.0	1.913164e-01	0.0
28	0.0	0.0	0.0	1e+00
29	-4.283737e-04	1.887799e-03	1.857876e-02	-1.638467e-01
30	-3.994248e-04	-1.298598e-03	1.740134e-01	2.686906e-02

31	-1.086978e-04	-5.324735e-04	2.018427e-01	-4.432381e-03
32	-2.067074e-04	-4.782434e-04	2.615591e-01	7.555245e-04
33	-1.865691e-05	-4.817452e-04	2.820227e-01	-1.478511e-04
34	-6.299328e-06	-3.788040e-04	2.858883e-01	4.049164e-05
35	4.735664e-06	-4.906313e-04	2.854844e-01	-1.395098e-05
36	0.0	0.0	2.849215e-01	0.0
37	0.0	0.0	0.0	1e+00
38	-7.937033e-04	-1.914637e-03	1.609532e-01	-1.638486e-01
39	-4.701302e-04	-2.580780e-04	3.114481e-01	2.687075e-02
40	-1.528478e-04	-7.987321e-04	3.765080e-01	-4.433557e-03
41	-5.256368e-05	-6.478469e-04	4.029606e-01	7.562430e-04
42	3.763590e-05	-4.847742e-04	4.011488e-01	-1.482409e-04
43	4.290054e-05	-3.932583e-04	3.898778e-01	4.068039e-05
44	2.739513e-05	-4.628084e-04	3.806761e-01	-1.402507e-05
45	0.0	0.0	3.768469e-01	0.0
46	0.0	0.0	0.0	1e+00
47	-1.629444e-03	9.065719e-04	4.239107e-01	-1.638478e-01
48	1.743014e-04	-1.578654e-03	4.998751e-01	2.686856e-02
49	-2.973099e-04	-1.067135e-03	6.011451e-01	-4.431695e-03
50	2.959970e-04	-6.323561e-04	5.709631e-01	7.550774e-04
51	1.522233e-04	-5.213909e-04	5.238312e-01	-1.476182e-04
52	1.233061e-04	-3.497325e-04	4.902916e-01	4.038569e-05
53	6.218657e-05	-4.089371e-04	4.676727e-01	-1.391170e-05
54	0.0	0.0	4.604967e-01	0.0
55	0.0	0.0	0.0	1e+00
56	-2.329357e-03	-3.422810e-03	3.035360e-01	-1.638532e-01
57	-2.320704e-03	-2.614862e-03	1.077133e+00	2.687324e-02
58	1.471789e-03	-8.513105e-04	8.695148e-01	-4.434813e-03
59	1.226168e-04	-7.138574e-04	7.280007e-01	7.569744e-04
60	3.314803e-04	-3.764616e-04	6.438067e-01	-1.486371e-04
61	1.760095e-04	-2.860325e-04	5.739185e-01	4.087178e-05
62	8.007267e-05	-3.036905e-04	5.403497e-01	-1.409971e-05
63	0.0	0.0	5.280463e-01	0.0
64	0.0	0.0	0.0	1e+00
65	-8.370939e-03	-1.358976e-02	2.621769e+00	-1.638581e-01
66	1.004390e-02	-5.978473e-04	1.552912e+00	2.686748e-02
67	-4.051999e-04	-8.748004e-04	1.046587e+00	-4.430209e-03
68	1.356747e-03	-3.767601e-04	8.935898e-01	7.541225e-04

69	5.682398e-04	-2.297423e-04	7.097439e-01	-1.470629e-04
70	3.044300e-04	-1.520635e-04	6.355466e-01	4.009788e-05
71	1.613550e-04	-1.661850e-04	5.866207e-01	-1.379273e-05
72	0.0	0.0	5.732381e-01	0.0
73	0.0	0.0	0.0	1e+00
74	0.0	0.0	5.118684e+00	-1.638515e-01
75	0.0	0.0	1.494344e+00	2.687271e-02
76	0.0	0.0	1.231971e+00	-4.435480e-03
77	0.0	0.0	9.133950e-01	7.576172e-04
78	0.0	0.0	7.505669e-01	-1.490509e-04
79	0.0	0.0	6.528921e-01	4.109640e-05
80	0.0	0.0	6.044457e-01	-1.419493e-05
81	0.0	0.0	5.900228e-01	0.0

The graph of allocation of temperature in the investigated area is shown on the following *fig. 12.10*.

### Temperature $T(x,y)$ Distribution:

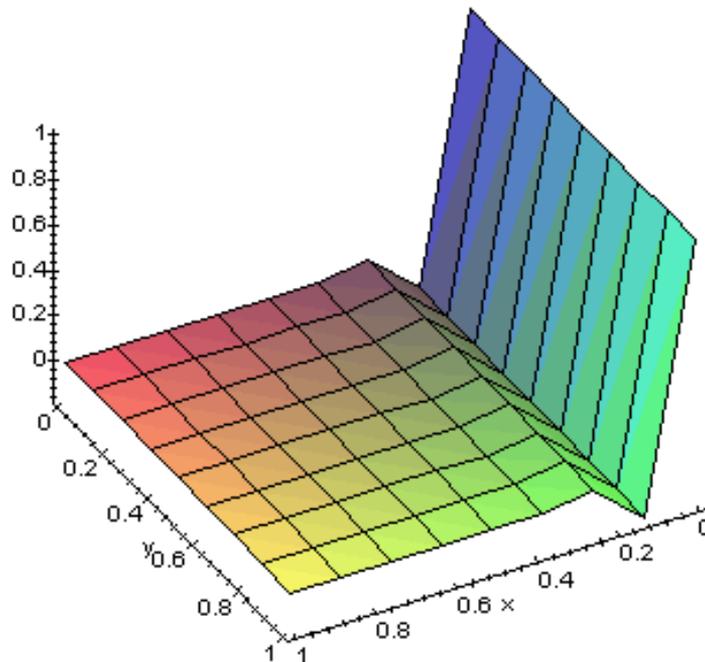


Fig. 12.10. Allocation of temperature in quadratic area;  $Pr=0.7$  and  $Gr=10^3$

While the graph of a *profile of velocity  $u_x$  at  $x=0.5$*  is shown on the *fig. 12.11*. The physical picture of a motion of a fluid in the considered area consists in that, that the layers of a fluid, heated at the left-hand wall, go up, while the layers of a fluid cooled at the right wall go down. In this connection in the area a circulation motion, which intermixes a fluid and transports heat from a heated wall to cold wall, arises.

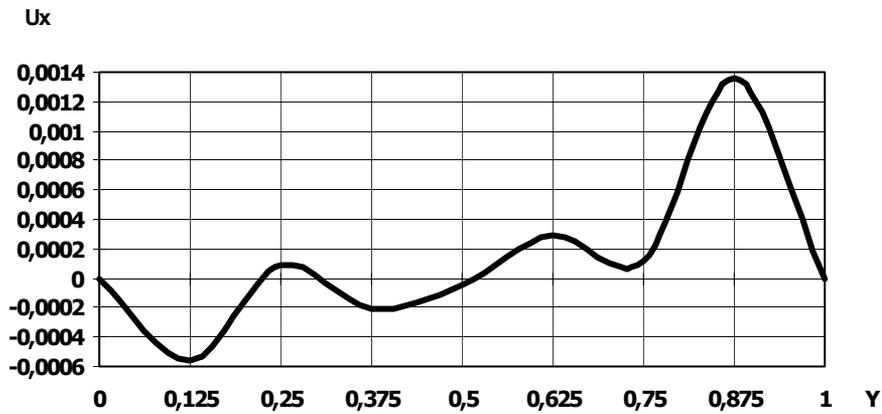


Fig. 12.11. The graph of profile of velocity  $u_x$  of a fluid at  $x=0.5$ .

The *Heat\_flow* program is intended for the solution of the equations of convective heat exchange of a fluid in an arbitrary flat contour. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in PROBLEMS directory of archive attached to the present book in datafiles *Mb9\_3\_new.mws*, *Mb9\_3.dat* and *Mb9\_3\_1.rez* accordingly.

## 12.4. Problem of heat exchange and and mass transfer for an incompressible fluid

The phenomena of heat exchange and mass transfer concern more common process called as a *variance*. *Variance* is being understood as change of amount of substance dissolved, for example, in a fluid. This phenomenon includes such processes as a *diffusion*, *convection*, *decay* etc. The ecological problems, in particular, include transport of harmful substances in fluids and gases (*air*). Owing to this and a whole series of other reasons the problem of heat exchange and mass transfer has rather wide practical application.

### 12.4.1. The calculated equations of heat exchange and mass transfer

The motion of a fluid and gas is described by the known equations of Navier-Stokes. The transport of heat is described by the *equation of energy*, while transport of mass - by the *mass transfer equation*. The system of the two-dimensional equations of Navier-Stokes for a convection, heat exchange and mass transfer in Bussinesk's approximation is considered. It is supposed, that the fluid dynamically and statically is not compressible, i.e. its density does not depend on pressure, but can depend on temperature and concentration of the weighed substance. The equations system of heat exchange and mass transfer, written in the dimensionless form, accepts the following general form:

$$\begin{aligned} \frac{\partial u_x}{\partial t} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_x}{\partial y} &= \frac{1}{Re} \left( \Delta u_x - \frac{\gamma}{x^2} u_x \right) - \frac{\partial p}{\partial x}; \\ \frac{\partial u_y}{\partial t} + u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} &= \frac{1}{Re} \Delta u_y - \frac{\partial p}{\partial y} + \frac{Gr}{Re^2} T + \frac{Gr_c}{Re^2} c; \\ \frac{\partial u_x}{\partial x} + \frac{\gamma}{x} u_x + \frac{\partial u_y}{\partial y} &= 0; \quad \frac{\partial T}{\partial t} + u_x \frac{\partial T}{\partial x} + u_y \frac{\partial T}{\partial y} = \frac{1}{Re Pr} \Delta T; \quad \frac{\partial c}{\partial t} + u_x \frac{\partial c}{\partial x} + u_y \frac{\partial c}{\partial y} = \frac{1}{Re Sc} \Delta c \end{aligned} \quad (12.52)$$

where  $u_x$ ,  $u_y$  - radial and axial components of velocity in a cylindrical frame ( $r=x$ ,  $z=y$ );  $\gamma = 0$ , using cartesian coordinates, and  $\gamma = 1$ , using cylindrical frame;  $p$  - pressure;  $T$  - temperature;  $c$  -

concentration of substance;  $\mathbf{Re}$  - the Reynolds number ( $\mathbf{Re} = \frac{\mathbf{vL}}{\nu}$ ,  $\nu$  - scale of velocity);  $L$  - scale of the length;  $\nu$  - kinematic viscosity;  $\mathbf{Pr}$  - Prandtl number ( $\mathbf{Pr} = \frac{\nu}{a}$ ,  $a = \frac{\lambda}{\rho c_p}$  - thermal diffusivity;  $\lambda$  - thermal conductivity;  $c_p$  - coefficient of a specific heat capacity at constant pressure);  $\mathbf{Gr}$  - the Grashof number ( $\mathbf{Gr} = \frac{g\beta_T L^3 \Delta T}{\nu^2}$ ,  $g$  - acceleration of gravity);  $\beta_T$  - coefficient of thermal change of density ( $\beta_T = -\frac{1}{\rho} \frac{\partial \rho}{\partial T}$ );  $\Delta T$  - difference of temperatures;  $\mathbf{Gr}_c$  - Grashof concentration number, ( $\mathbf{Gr}_c = \frac{g\beta_c L^3 \Delta c}{\nu^2}$ ,  $\beta_c$  - coefficient of concentration change of density,  $\beta_c = -\frac{1}{\rho} \left( \frac{\partial \rho}{\partial c} \right)_{p,T}$ );  $\mathbf{Sc}$  - Schmidt's number or Prandtl diffusion number ( $\mathbf{Sc} = \frac{\nu}{D}$ ,  $D$  - diffusion constant). The Laplace operator has the following well known view:

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\gamma}{x} \frac{\partial}{\partial x} + \frac{\partial^2}{\partial z^2} \quad (12.53)$$

A mode of transport of heat (*mass*) by molecular processes of a thermal conduction (*diffusion*) are implemented in a fixed fluid. These modes are *asymptotic* for system (12.52) under conditions  $\mathbf{Re} \rightarrow 0$ ,  $\mathbf{Gr} \rightarrow 0$  and  $\mathbf{Gr}_c \rightarrow 0$ . Under condition  $\mathbf{Gr} = \mathbf{Gr}_c$  the equations (12.52) represent the equations of Navier-Stokes for a *homogeneous incompressible fluid*. Last two equations (12.52) in this case describe a transport of heat and mass by a moving fluid; in addition, the process of heat exchange and mass transfer does not influence the motion of a fluid. The system (12.52) describes a thermal and concentration natural gravitational convection. The intensity of a thermal and concentration convection is defined by the Grashof number. The essential value has the Prandtl number, representing the ratio of thickness of dynamic and thermal boundary layers in a fluid. During a natural convection a typical velocity  $\mathbf{v}$  is absent. As a scale of velocity  $\mathbf{v}$  for the system (12.52) the quantity  $\mathbf{v}/L$  can be taken. In addition, the Reynolds number is necessary to suppose by equal to one, i.e.  $\mathbf{Re}=1$ .

The method of correction of velocity (see subsection 12.3) is applied to the equations system (12.52), while the equations system is solved by means of the FEM, for that the *four-nodal isoparametric finite element* is used; in this element a velocity, pressure, temperature and concentration of substance are approximated by the following relations:

$$\varphi(r, s) = \sum_{i=1}^4 \mathbf{N}_i(r, s) \varphi_i = [\mathbf{N}(r, s)]\{\varphi\}, \text{ where } \varphi \in (u_x, u_y, p, T, c) \quad (12.54)$$

where  $\mathbf{N}_i$  - shape functions. By applying the Galerkin's procedure, the equations system (12.52) accepts the following common view, namely:

$$\begin{aligned} [\mathbf{M}]\{\mathbf{U}_{x,n}\} &= [\mathbf{M}]\{\mathbf{U}_{x,n}\} + \tau\{[\mathbf{F}_{ux}] - ([\mathbf{A}] + [\mathbf{K}_{ux}])\{\mathbf{U}_{x,n}\}\}; \\ [\mathbf{M}]\{\mathbf{U}_{y,n}\} &= [\mathbf{M}]\{\mathbf{U}_{y,n}\} + \tau\{[\mathbf{F}_{uy}] + [\mathbf{F}_{uyt}] + [\mathbf{F}_{uyc}] - ([\mathbf{A}] + [\mathbf{K}_{uy}])\{\mathbf{U}_{y,n}\}\}; [\mathbf{K}_p]\{\mathbf{P}_{n+1}\} = [\mathbf{F}_{p_1}] - [\mathbf{F}_{p_2}]; \\ [\mathbf{M}]\{\mathbf{U}_{x,n+1}\} &= [\mathbf{M}]\{\mathbf{U}_{x,n}\} - \tau\{[\mathbf{R}_{ux}]\}; [\mathbf{M}]\{\mathbf{U}_{y,n+1}\} = [\mathbf{M}]\{\mathbf{U}_{y,n}\} - \tau\{[\mathbf{R}_{uy}]\}; \\ [\mathbf{M}]\{\mathbf{T}_{n+1}\} &= [\mathbf{M}]\{\mathbf{T}_n\} + \tau\{[\mathbf{F}_T] - ([\mathbf{A}] + [\mathbf{K}_T])\{\mathbf{T}_n\}\}; [\mathbf{M}]\{\mathbf{C}_{n+1}\} = [\mathbf{M}]\{\mathbf{C}_n\} + \tau\{[\mathbf{F}_c] - ([\mathbf{A}] + [\mathbf{K}_c])\{\mathbf{C}_n\}\}, \end{aligned} \quad (12.55)$$

$$\text{where: } \{F_{uyc}\} = \sum_{e=1}^{NE} \{f_{uyc}^{(e)}\}, \quad \{f_{uyc}^{(e)}\} = \frac{Gr_c}{Re^2} \int_{A^{(e)}} [N]^T [N] \{c\} dA; \quad \{F_c\} = \sum_{e=1}^{NE} \{f_c^{(e)}\},$$

$$\{f_c^{(e)}\} = \frac{1}{Re Sc} \int_{\Gamma^{(e)}} [N]^T \left( \frac{\partial c}{\partial x} I_x + \frac{\partial c}{\partial y} I_y \right) x^\gamma d\Gamma; \quad \{K_c\} = \sum_{e=1}^{NE} \{k_c^{(e)}\}, \quad (12.56)$$

$$\{k_c^{(e)}\} = \frac{1}{Re Sc} \int_{A^{(e)}} \left( \left[ \frac{\partial N}{\partial x} \right]^T \left[ \frac{\partial N}{\partial x} \right] + \left[ \frac{\partial N}{\partial y} \right]^T \left[ \frac{\partial N}{\partial y} \right] \right) dA; \quad dA = x^\gamma dx dy;$$

$n$  – number of an integration step. The given equations system is solved with the next *boundary* and *initial* conditions:

$$u_x|_{\Gamma} = u_{x_{r_0}}, \quad u_y|_{\Gamma} = u_{y_{r_0}}, \quad p|_{\Gamma} = p_{r_0}, \quad T|_{\Gamma} = T_{r_0}, \quad c|_{\Gamma} = c_{r_0};$$

$$\frac{\partial u_x}{\partial x} \Big|_{\Gamma} = -q_{ux}, \quad \frac{\partial u_y}{\partial y} \Big|_{\Gamma} = -q_{uy}, \quad \frac{\partial p}{\partial x} \Big|_{\Gamma} = -q_{px}, \quad \frac{\partial p}{\partial y} \Big|_{\Gamma} = -q_{py};$$

$$\frac{\partial T}{\partial x} \Big|_{\Gamma} = -q_{Tx}, \quad \frac{\partial T}{\partial y} \Big|_{\Gamma} = -q_{Ty}, \quad \frac{\partial c}{\partial x} \Big|_{\Gamma} = -q_{cx}, \quad \frac{\partial c}{\partial y} \Big|_{\Gamma} = -q_{cy}; \quad (12.57)$$

$$u_x|_{t=0} = u_{x_0}(x, y), \quad u_y|_{t=0} = u_{y_0}(x, y); \quad p|_{t=0} = p_0(x, y), \quad T|_{t=0} = T_0(x, y), \quad c|_{t=0} = c_0(x, y)$$

where  $q_{ux}, q_{uy}$  – stream of a fluid;  $q_{px}, q_{py}$  – components of a pressure gradient;  $q_{Tx}, q_{Ty}$  – stream of a heat;  $q_{cx}, q_{cy}$  – diffusion stream. The equations of a convection in the Brussinek's approximation differ by the specificity in view of considerable influence of fields of friction, temperature and concentration of substance. Nonstationarity of current, conditioned by its instability, is discovered for the given class of motions at smaller values of the Reynolds number, than in case of a motion of an isothermal fluid. The information about these modes holds in the non-stationary equations of Navier-Stokes.

In the equations system of Navier-Stokes there is a small parameter, which essentially influences a smoothness of the solution. That is linked with occurrence at a surface at increase of number  $Re$  of a boundary layer, thickness of which is proportional to quantity  $(Re)^{-1/2}$ . The equations system of Navier-Stokes is *non-linear*. This nonlinearity is typical for systems of a hydrodynamic type and is conditioned (*in case of an incompressible fluid*) by inertial components in the equations of a momentum. At large Reynolds numbers in a moving medium the composite spatio-temporal structures arise; the motion becomes indigested, chaotic. For the given reason in the present book some problems for the solution of the equations of Navier-Stokes are represented.

#### 12.4.2. Input data for the solution of the problem

For the solution of the equations system describing a heat exchange and mass transfer in some plane, the *Heat\_mass\_flow* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable  $F$ , necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely.

In the *first* line the next parameters are coded: the number of finite elements (*parameter nelem*), number of nodes (*npoint*), the number of Dirichlet's boundary conditions for *velocity* in direction of axes **X** and **Y** (*nbundu*, *nbondv*), for *pressure* (*nbondp*), for *temperature* (*nbondt*) and for *concentration of substance* (*nbondc*). Into the *second* line the number of Neumann's boundary conditions for *velocity* in direction of axes **X** and **Y** (*ngradu*, *ngradv*), for *pressure* (*ngradp*), for *temperature* (*ngradt*) and for *concentration of substance* (*ngradc*) are recorded.

In the *third* line the next parameters are coded: a print code of intermediate results (*kprint*), a type of a problem (*ngama*), the Reynolds number (*reinold*), the Prandtl number (*prandl*), the Grashof number (*grashof*) and Schmidt number (*sc*) are indicated. If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out. If the problem is being solved in *Cartesian* frames, then *ngama*=0, otherwise - in *cylindrical* frames is solved.

In the *fourth* line, a code of the solution of system of the algebraic equations (*ksolve*), the number of iterations (*niter*), which is necessary for the solution of an equations system, and a precision of the solution (*toler*). if parameter *ksolve*=0, then for the solution, the *'solve'* function of the *Maple*-language is used; otherwise, the equations system by the method of *conjugate gradients* is being solved. In the *fifth* line, the number of integration steps (*ntime*) and also size of an integration step (*dtime*) are coded.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop**(*nelem*, *nnode*), where: *nnode* - number of nodes of a finite element, i.e. *nnode* = 4} are coded. After of array **Mtop** the elements of **Lbondu**(*nbundu*) array are coded line by line. Into each line of the file the line number and element of **Lbondu** array are recorded. Into the **Lbondu** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *velocity* in direction of axis **X**, are coded.

After of array **Lbondu** the elements of **Lbondv**(*nbondv*) array are coded line by line. Into each line of the file the line number and element of **Lbondv** array are recorded. Into the **Lbondv** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *velocity* in direction of axis **Y**, are coded. After of array **Lbondv** the elements of **Lbondp**(*nbondp*) array are coded line by line. Into each line of the file the line number and element of **Lbondp** array are recorded. Into the **Lbondp** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *pressure*, are coded.

After of array **Lbondp** the elements of **Lbondt**(*nbondt*) array are coded line by line. Into each line of the file the line number and element of **Lbondt** array are recorded. Into the **Lbondt** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *temperature*, are coded. After of array **Lbondt** the elements of **Lbondc**(*nbondc*) array are coded line by line. Into each line of the file the line number and element of **Lbondc** array are recorded. Into the **Lbondc** array the numbers of nodes, in which are known Dirichlet's boundary conditions for *concentration of substance* in a fluid are coded.

After of array **Lbondc** the elements of **Lgradu**(*ngradu*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradu** array are recorded. In this array the next parameters are coded: the line number of **Lgradu** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *velocity* in direction of axis **X** are preset.

After of array **Lgradu** the elements of **Lgradv**(*ngradv*, 3) array are coded line by line. Into each line of the file the line number and element of **Lgradv** array are recorded. In this array the next parameters are coded: the line number of **Lgradv** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *velocity* in direction of axis **Y** are preset.

After of array **Lbondv** the elements of **Lgradp(ngradp, 3)** array are coded line by line. Into each line of the file the line number and element of **Lgradp** array are recorded. In this array the next parameters are coded: the line number of **Lgradp** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *pressure* are given.

After of array **Lbondp** the elements of **Lgradt(ngradt, 3)** array are coded line by line. Into each line of the file the line number and element of **Lgradt** array are recorded. In this array the next parameters are coded: the line number of **Lgradt** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *temperature* are given.

After of array **Lbondt** the elements of **Lgradc(ngradc, 3)** array are coded line by line. Into each line of the file the line number and element of **Lgradc** array are recorded. In this array the next parameters are coded: the line number of **Lgradc** array, the number of a finite element and also two numbers of nodes which are on a side of the finite element, laying on a surface, where boundary conditions of Neumann for *concentration of substance* in a fluid are given.

Behind of array **Lgradc** into the data file the elements of **Coord(npoin, ndime)** array, where *ndime* - dimensionality of the problem (*ndime*=2), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, into the *second* column of the **Coord** array the *y*-coordinates of nodes of finite elements are coded. Into each line of the file the number of a node, and also its (*x*, *y*)-coordinates are recorded. Behind of array **Coord** the elements of **Bondu(nbondu)** array are recorded line by line. Into each line of the datafile a line number of **Bondu** array and a value of *velocity* in direction of axis **X** are recorded. The lines of **Bondu** array should strictly correspond to the lines of **Lbondu** array.

Behind of array **Bondu** the elements of **Bondv(nbondv)** array are recorded line by line. Into each line of the datafile a line number of **Bondv** array and a value of *velocity* in direction of axis **Y** are recorded. The lines of **Bondv** array should strictly correspond to the lines of **Lbondv** array. Behind of array **Bondv** the elements of **Bondp(nbondp)** array are recorded line by line. Into each line of the datafile a line number of **Bondp** array and a value of *pressure* are recorded. The lines of **Bondp** array should strictly correspond to the lines of **Lbondp** array. Behind of array **Bondp** the elements of **Bondt(nbondt)** array are recorded line by line. Into each line of the datafile a line number of **Bondt** array and a value of *temperature* are recorded. The lines of **Bondt** array should strictly correspond to the lines of **Lbondt** array. Behind of array **Bondt** the elements of **Bondc(nbondc)** array are recorded line by line. Into each line of the datafile a line number of **Bondc** array and a value of *concentration of substance* in a fluid are recorded. The lines of **Bondc** array should strictly correspond to the lines of **Lbondc** array.

Behind of array **Bondt** the elements of **Gradu(ngradu, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradu** array and also values of derivatives  $\frac{\partial u_x}{\partial x}$  and  $\frac{\partial u_x}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradu** array should strictly correspond to the lines of **Lgradu** array. Behind of array **Gradu** the elements of **Gradv(ngradv, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradv** array and also values of derivatives  $\frac{\partial u_y}{\partial x}$  and  $\frac{\partial u_y}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradv** array should strictly correspond to the lines of **Lgradv** array.

Behind of array **Gradv** the elements of **Gradp(ngradp, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradp** array and also values of derivatives  $\frac{\partial p}{\partial x}$  and  $\frac{\partial p}{\partial y}$

$\frac{\partial p}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradp** array should strictly correspond to the lines of **Lgradp** array. After of array **Gradp** the elements of **Gradt(ngradt, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradt** array and also values of derivatives  $\frac{\partial T}{\partial x}$  and  $\frac{\partial T}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradt** array should strictly correspond to the lines of **Lgradt** array. At last, after of array **Gradt** the elements of **Gradc(ngradc, 2)** array are recorded line by line. Into each line of the datafile are coded a line number of **Gradc** array and also values of derivatives  $\frac{\partial c}{\partial x}$  and  $\frac{\partial c}{\partial y}$  in the nodes, which are on a surface. The lines of **Gradc** array should strictly correspond to the lines of **Lgradc** array. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*

*nelem, npoin, nbondu, nbondv, nbondp, nbondt, nbondc, ngradu, ngradv, ngradp, ngradt, ngradc, kprint, ngama, reynold, prandl, grashof, sc, ksolve, niter, toler*

Text line \*

Arrays **Mtop(nelem, nnode)**

Text line \*

Array **Lbondu(nbondu)**

Text line \*

Array **Lbondv(nbondv)**

Text line \*

Array **Lbondp(nbondp)**

Text line \*

Array **Lbondt(nbondt)**

Text line \*

Array **Lbondc(nbondc)**

Text line \*

Array **Lgradu(ngradu, 3)**

Text line \*

Array **Lgradv(ngradv, 3)**

Text line \*

Array **Lgradp(ngradp, 3)**

Text line \*

Array **Lgradt(ngradt, 3)**

Text line \*

Array **Lgradc(ngradc, 3)**

Text line \*

Array **Coord(npoin, ndime)**

Text line \*

Array **Bondu(nbondu)**

Text line \*

Array **Bondv(nbondv)**

Text line \*

Array **Bondp(nbondp)**

Text line \*

Array *Bondt*(*nbondt*)  
 Text line \*  
 Array *Bondc*(*nbondc*)  
 Text line \*  
 Array *Gradu*(*ngradu*, 2)  
 Text line \*  
 Array *Gradv*(*ngradv*, 2)  
 Text line \*  
 Array *Gradp*(*ngradp*, 2)  
 Text line \*  
 Array *Gradt*(*ngradt*, 2)  
 Text line \*  
 Array *Gradc*(*ngradc*, 2)

### 12.4.3. Brief description of the Heat\_mass\_flow program solving the problem

The *Heat\_mass\_flow* program was programmed on the *Maple*-language; it consists of the basic program and 51 procedures. All procedures can be divided into three groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates the values of *velocities* in direction of axes **X** and **Y**, and also of *pressure*, *temperature* and *concentration of substance* in a fluid in nodes of the finite elements. The calculation results are output on the monitor and are recorded into files, for that to variable *file\_rez1* must be ascribed qualifier of the target file. In this file the values of *velocities* in direction of axes **X** and **Y**, *pressure*, *temperature* and *concentration of substance* in a fluid in nodes of the finite elements are recorded.

### 12.4.4. An example of use of the Maple-program Heat\_mass\_flow

A problem of *heat exchange and mass transfer* in a closed square area of size (1.0x1.0) at isothermal side boundaries is considered. On the left boundary of the area a *concentration of substance* is preset. The *boundary conditions* for this problem are given as follows:

$$x = 0, 0 \leq y \leq 1, u_x = u_y = 0, p = 0, T = 1; x = 1, 0 \leq y \leq 1; u_x = u_y = 0, T = 0; y = 0 \text{ and } y = 1, 0 \leq x \leq 1, u_x = u_y = 0, \frac{\partial T}{\partial y} = 0$$

The calculated scheme of the closed square area is represented on the fig. 12.12.

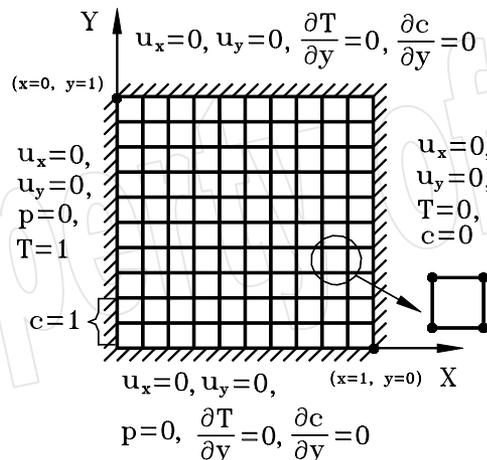


Fig. 12.12. The calculated scheme and boundary conditions in the square area

*Input data for the test example: nelem=100, npoin=121, nbundu=40, nbondv=40, nbondp=21, nbondt = 22, nbondc=14, ngradu=1, ngradv=1, ngradp=1, ngradt=1, ngradc=1, kprint=0, ngama=0, ksolve=1, niter=50, toler=10<sup>-6</sup>, ntime=9, dtime=10<sup>-3</sup> s, Reynold number reinold=1, Prandtl number prandl = 0.7, Grashof number grashof=10<sup>3</sup>, Schmidt number sc=10.*

***Results of calculation over the test example:***

*The following table represents the values of velocities (ux, uy), pressure (press), temperature (temp) and concentration of substance (mass) in nodes (node) of finite elements, namely:*

<i>node</i>	<i>ux</i>	<i>uy</i>	<i>press</i>	<i>temp</i>	<i>mass</i>
1	0.0	0.0	0.0	1e+00	1e+00
2	0.0	0.0	0.0	-9.317124e-02	-3.791072e+01
3	0.0	0.0	0.0	1.491395e-02	4.985332e+01
4	0.0	0.0	0.0	-9.215330e-03	-4.347536e+01
5	0.0	0.0	0.0	8.527676e-03	3.076323e+01
6	0.0	0.0	0.0	-7.155301e-03	-1.912125e+01
7	0.0	0.0	0.0	5.049267e-03	1.109107e+01
8	0.0	0.0	0.0	-2.582940e-03	-6.692747e+00
9	0.0	0.0	0.0	0.0	5.322389e+00
10	0.0	0.0	0.0	1e+00	1e+00
11	5.019274e-03	2.303857e-03	-2.710808e+00	-9.326110e-02	4.036899e+01
12	-1.925422e-02	2.064703e-03	-3.462044e-02	1.501563e-02	-5.286489e+01
13	4.187939e-03	-9.687478e-04	1.152727e-01	-9.263057e-03	4.605281e+01
14	-1.318200e-03	-1.576678e-03	1.247392e-01	8.541922e-03	-3.263314e+01
15	-2.857219e-04	-1.973074e-03	1.724614e-01	-7.152862e-03	2.033675e+01
16	-8.016058e-05	-1.941933e-03	2.306690e-01	5.043283e-03	-1.183295e+01
17	2.385585e-04	-2.690750e-03	1.878167e-01	-2.579082e-03	7.162139e+00
18	0.0	0.0	1.942479e-01	0.0	-5.704291e+00
19	0.0	0.0	0.0	1e+00	1e+00
20	1.114307e-03	-1.927849e-02	4.192562e-01	-9.328567e-02	-4.791256e+01
21	2.039825e-03	-1.790160e-03	-2.693745e-01	1.504713e-02	6.120290e+01
22	-4.158778e-03	-1.020943e-03	2.034250e-01	-9.314816e-03	-5.296043e+01
23	1.496483e-05	-1.741480e-03	3.374778e-01	8.587661e-03	3.761701e+01
24	-3.862537e-04	-1.658666e-03	3.899836e-01	-7.186779e-03	-2.364109e+01
25	-1.007486e-04	-1.244104e-03	4.052500e-01	5.063969e-03	1.394201e+01
26	-3.003393e-05	-2.078298e-03	4.240001e-01	-2.588549e-03	-8.582820e+00
27	0.0	0.0	4.144592e-01	0.0	6.902874e+00
28	0.0	0.0	0.0	1e+00	-6.689929e+01
29	-1.989985e-03	7.286478e-03	1.026371e-01	-9.306482e-02	8.621988e+01
30	-1.284107e-03	-4.854998e-03	3.801588e-01	1.475795e-02	-8.243537e+01

31	-5.620926e-04	-2.591047e-03	4.331663e-01	-9.074910e-03	6.437773e+01
32	-8.539675e-04	-1.980691e-03	5.654063e-01	8.429206e-03	-4.362222e+01
33	-1.077085e-04	-1.960720e-03	6.046161e-01	-7.094613e-03	2.675553e+01
34	-5.372561e-05	-1.735589e-03	6.218446e-01	5.016447e-03	-1.556992e+01
35	2.210287e-05	-2.050918e-03	6.175881e-01	-2.568961e-03	9.515801e+00
36	0.0	0.0	6.203780e-01	0.0	-7.633846e+00
37	0.0	0.0	0.0	1e+00	8.805819e+01
38	-3.423702e-03	-7.740872e-03	3.306125e-01	-9.363225e-02	-9.363756e+01
39	-2.066396e-03	-1.461313e-03	6.754205e-01	1.544415e-02	8.180913e+01
40	-5.742514e-04	-3.032811e-03	8.036078e-01	-9.608501e-03	-6.068009e+01
41	-2.914924e-04	-2.828562e-03	8.601200e-01	8.763745e-03	3.970380e+01
42	1.151367e-04	-2.130647e-03	8.667893e-01	-7.277300e-03	-2.369353e+01
43	1.550480e-04	-1.641772e-03	8.441279e-01	5.105136e-03	1.345489e+01
44	8.545289e-05	-2.114281e-03	8.309072e-01	-2.603825e-03	-8.038217e+00
45	0.0	0.0	8.235053e-01	0.0	6.373941e+00
46	0.0	0.0	0.0	1e+00	-7.938233e+01
47	-7.058072e-03	3.987046e-03	9.230165e-01	-9.260848e-02	7.920714e+01
48	9.247907e-04	-5.804675e-03	1.068007e+00	1.417399e-02	-6.607656e+01
49	-1.236597e-03	-5.044955e-03	1.256263e+00	-8.615124e-03	4.727893e+01
50	1.031019e-03	-2.538755e-03	1.227154e+00	8.137676e-03	-3.000784e+01
51	6.814282e-04	-2.173204e-03	1.124466e+00	-6.935074e-03	1.741995e+01
52	4.104644e-04	-1.608242e-03	1.062805e+00	4.939051e-03	-9.630572e+00
53	2.097150e-04	-1.783260e-03	1.025205e+00	-2.538601e-03	5.604975e+00
54	0.0	0.0	1.010452e+00	0.0	-4.383821e+00
55	0.0	0.0	0.0	1e+00	6.089225e+01
56	-9.004137e-03	-1.335474e-02	5.450565e-01	-9.424803e-02	-5.886721e+01
57	-1.040903e-02	-1.178475e-02	2.194914e+00	1.621222e-02	4.753145e+01
58	5.535657e-03	-3.000326e-03	1.895332e+00	-1.023774e-02	-3.297623e+01
59	1.076603e-03	-3.104438e-03	1.532254e+00	9.181065e-03	2.033316e+01
60	1.018871e-03	-1.647380e-03	1.379691e+00	-7.515719e-03	-1.148334e+01
61	6.900273e-04	-1.164938e-03	1.254181e+00	5.225121e-03	6.178200e+00
62	3.233888e-04	-1.503813e-03	1.185319e+00	-2.652145e-03	-3.500670e+00
63	0.0	0.0	1.172644e+00	0.0	2.699147e+00
64	0.0	0.0	0.0	1e+00	-4.587978e+01
65	-3.665749e-02	-5.311809e-02	5.195907e+00	-9.207459e-02	4.351329e+01
66	3.906254e-02	-5.654934e-03	3.509720e+00	1.334657e-02	-3.424881e+01
67	1.919964e-03	-3.391456e-03	2.107445e+00	-7.896947e-03	2.312910e+01
68	3.570260e-03	-1.023181e-03	1.894428e+00	7.636031e-03	-1.388679e+01

69	2.839046e-03	-1.570872e-03	1.549609e+00	-6.632721e-03	7.642535e+00
70	1.178892e-03	-2.034477e-04	1.352690e+00	4.779018e-03	-4.007135e+00
71	4.227927e-05	-1.102713e-03	1.320549e+00	-2.471584e-03	2.213046e+00
72	0.0	0.0	1.292373e+00	0.0	-1.682793e+00
73	0.0	0.0	0.0	1e+00	4.035986e+01
74	0.0	0.0	9.954501e+00	-9.464333e-02	-3.799511e+01
75	0.0	0.0	3.628505e+00	1.676672e-02	2.956771e+01
76	0.0	0.0	2.585321e+00	-1.073538e-02	-1.971615e+01
77	0.0	0.0	1.833132e+00	9.536598e-03	1.168676e+01
78	0.0	0.0	1.706424e+00	-7.734636e-03	-6.350658e+00
79	0.0	0.0	1.350511e+00	5.343095e-03	3.287350e+00
80	0.0	0.0	1.379955e+00	-2.702181e-03	-1.792176e+00
81	0.0	0.0	1.365997e+00	0.0	1.353072e+00

The graph of *allocation of concentration of substance* in the considered area is shown on the *fig. 12.13*.

### Mass Concentration $C(x,y)$ Distribution:

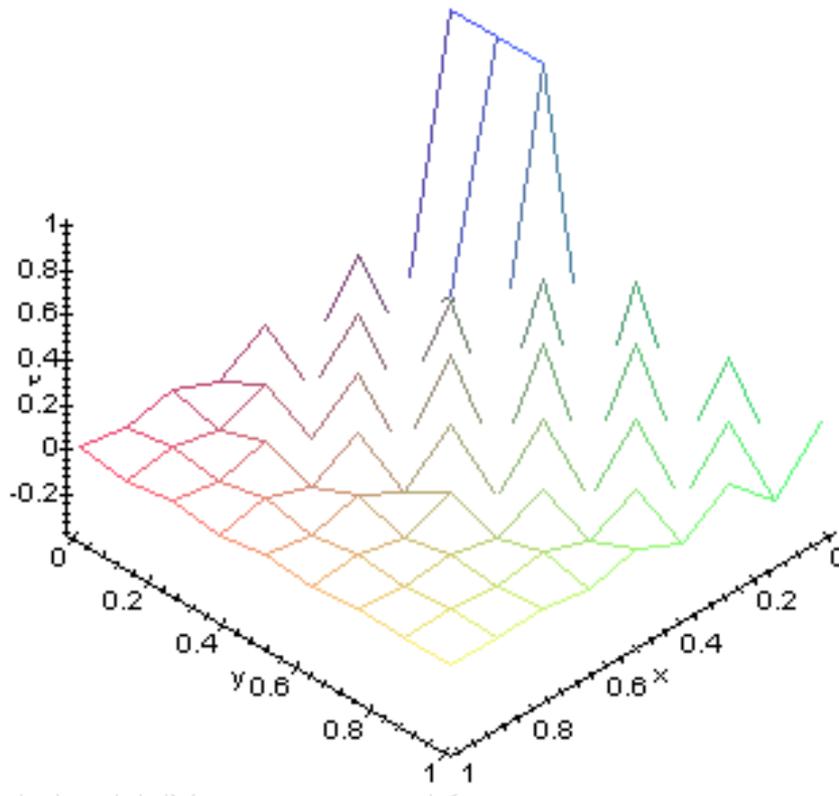


Fig. 12.13. Allocation of concentration of substance in quadratic area;  $Pr=0.7$ ,  $Gr=10^3$ ,  $Gr_c=10^3$  and  $Sc=0.5$ .

While the graph of a *profile of velocity  $u_x$*  at  $x=0.5$  is shown on the *fig. 12.14*.

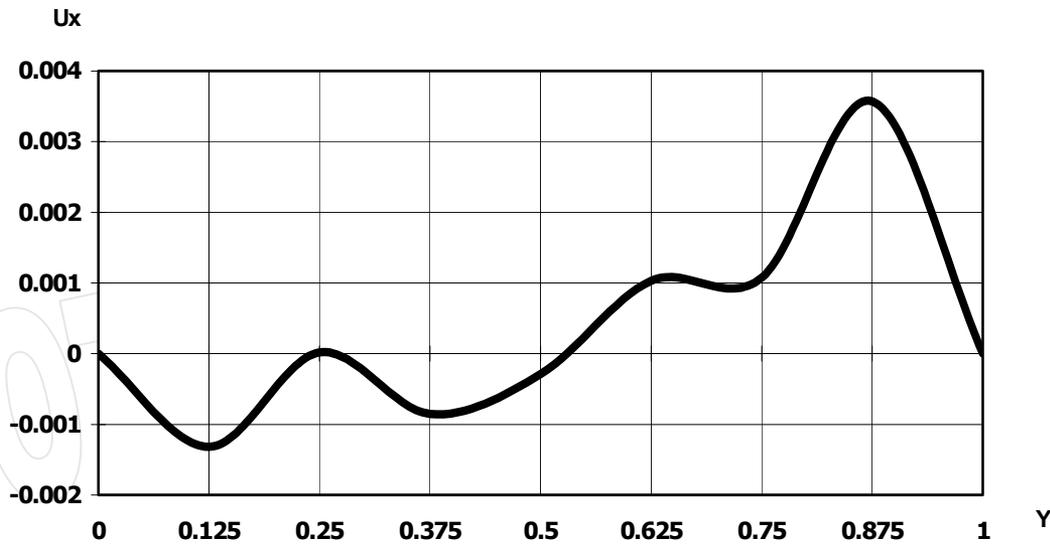


Fig. 12.14. The graph of profile of velocity  $u_x$  of a fluid at  $x=0.5$ .

The physical picture of a motion of a fluid in the considered area consists in that, that the layers of a fluid, heated at the left-hand wall, go up, while the layers of a fluid cooled at the right wall go down. In this connection in the area a circulation motion, which intermixs a fluid and transports heat from a heated wall to cold wall, arises.

The *Heat\_mass\_flow* program is intended for the solution of the equations of heat exchange and mass transfer of an incompressible fluid in an arbitrary flat contour. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb9\_4\_new.mws*, *Mb9\_4.dat* and *Mb9\_4\_1.rez* accordingly.

## 12.5. Models of motion of a fluid in a hydraulic system

The modern hydraulic systems include the various functional devices such as: hydraulic pumps, drives, electrical distributors, valves, hydro-accumulators, filters, pipelines etc. The processes happening in these systems, are quickly shifting, i.e. velocity and pressure quickly vary in a time. The hydraulic medium can represent an mixture containing a gas (*for example, air*). The presence of gas in a fluid sharply decreases velocity of distribution of a sound in such mixture and, accordingly, dynamic characteristics of processes, happening in it, have changed. The calculation of physical processes in hydraulic systems depending on properties of the system itself has the important applied value.

### 12.5.1. Motion of a compressible fluid in a hydraulic system

The motion of a compressible fluid in a hydraulic pipeline is supposed by one-dimensional and unsteady, i.e. all local velocities are considered equal to medial velocity and depend on a time. The pressure also is considered identical in all points of a clear area and depends on longitudinal coordinate of the pipeline and a time. Such motion of a fluid is characterized by origin of a wave of boosted and low pressure, which propagates from a place of change of pressure and strain of walls of the pipeline. A *fluid* represents an mixture containing a gas (*air*). The presence of gas in a fluid sharply decreases velocity of distribution of a sound, which for a case of air generally can be calculated according to the following formula:

$$c = \sqrt{\frac{\frac{K}{\rho}}{1 + \left(\frac{K}{E}\right)\left(\frac{d}{e}\right) + \left(\frac{\epsilon p_a R T}{K_a}\right)\left(\frac{K}{K_a} - 1\right)}} \quad (12.58)$$

where:  $\rho$ ,  $\rho_a$  – density of a fluid and air accordingly;  $K$ ,  $K_a$  – module of volumetric elasticity of a fluid and air for adiabatic process accordingly ( $K_a = \gamma p$ );  $d$ ,  $e$  – diameter and thickness of a pipeline accordingly;  $R$  – gas constant;  $T$  – temperature;  $\epsilon$  – relative amount of air in a fluid. Then the equation of motion of a viscous compressible fluid in the stand-pipe accepts the following form [50]:

$$\frac{1}{\rho} \frac{\partial p}{\partial x} + \frac{\partial u}{\partial x} + u \frac{\partial u}{\partial x} + \frac{f(u) u |u|}{2d} + g \sin \alpha = 0, \quad (12.59)$$

where:  $u$  – velocity of a fluid;  $\alpha$  – inclination angle of the pipeline to the horizontal plane;  $f$  – loss coefficient of pressure along the pipeline, namely:

$$f = \begin{cases} \frac{75}{Re}, & \text{when } Re \leq 2300; \\ 0,3164 \cdot Re^{-0,25}, & \text{when } > 2300, \end{cases} \quad (12.60)$$

where  $Re$  – Reynolds number ( $Re = \frac{ud}{\nu}$ ,  $\nu$  – kinematic viscosity of a fluid). Then the continuity equation of a motion of a fluid accepts the following form [50], namely:

$$\frac{\partial p}{\partial t} + u \frac{\partial p}{\partial x} + c^2 \rho \frac{\partial u}{\partial x} = 0 \quad (12.61)$$

As a rule, the differential equations of motion of a fluid in pipelines are solved by method of the characteristics [54-56,128]. The equations system (12.59) and (12.60) are the equations system in partial derivatives of a hyperbolic type, i.e. it has two real characteristics passing via each point of plane  $X$  and  $t$  (a time). The characteristics – curves, on which exist the algebraic equations or ordinary differential equations linking values of the sought functions  $u$  (velocity) and  $p$  (pressure). All length of a hydraulic pipeline is divided by elements of length  $\Delta x$ . The unknown variables are a velocity and pressure of a fluid at a moment  $t + \tau$  which are determined over values of these parameters at a moment  $t$  (fig. 12.15).

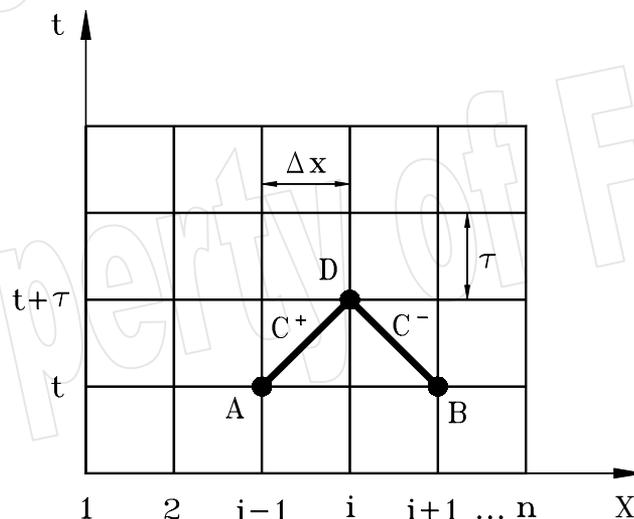


Fig. 12.15. The scheme of determination of parameters of a point  $D$  by method of the characteristics

Pressure and velocity in the point **D** at a moment  $\mathbf{t} + \tau$  are determined from the system of the nonlinear algebraic equations of the following form:

$$C^+ : \Phi_1 = p_D - p_A + B(u_D - u_A) + \frac{G}{8}[f(u_A) + f(u_D)] |u_A + u_D| (u_A + u_D) + g \sin \alpha = 0; \quad (12.62)$$

$$C^- : \Phi_2 = p_D - p_B - B(u_D - u_B) - \frac{G}{8}[f(u_B) + f(u_D)] |u_B + u_D| (u_B + u_D) - g \sin \alpha = 0 \quad (12.63)$$

where  $G = \frac{\rho \Delta x}{2d}$ ;  $B = \rho c$ . The system of the nonlinear algebraic equations (12.62) and (12.63) is solved by the Newton method, which in a matrix form has the following form:

$$[J]_i \{\Delta Y\}_i = -\{\Phi\}_i, \text{ where } \{\Delta Y\}_i^T = [\Delta p_i, \Delta u_i]; \quad \{\Delta \Phi\}_i^T = [\Phi_{1i}, \Phi_{2i}]; \quad (12.64)$$

$i$  - number of iteration;  $[J]$  - Jacobi matrix, namely:  $[J] = \begin{bmatrix} \frac{\partial \Phi_1}{\partial p_D} & \frac{\partial \Phi_1}{\partial u_D} \\ \frac{\partial \Phi_2}{\partial p_D} & \frac{\partial \Phi_2}{\partial u_D} \end{bmatrix}; \quad \frac{\partial \Phi_1}{\partial p_D} = 1;$

$$\frac{\partial \Phi_1}{\partial u_D} = B + \frac{G}{8} \left\{ \frac{\partial f}{\partial u_D} |u_A + u_D| (u_A + u_D) + 2[f(u_A) + f(u_D)] |u_A + u_D| \right\}; \quad \frac{\partial \Phi_2}{\partial p_D} = 1;$$

$$\frac{\partial \Phi_2}{\partial u_D} = -B - \frac{G}{8} \left\{ \frac{\partial f}{\partial u_D} |u_B + u_D| (u_B + u_D) + 2[f(u_B) + f(u_D)] |u_B + u_D| \right\}; \quad (12.65)$$

$$\frac{\partial f}{\partial u_D} = \frac{\partial f}{\partial Re} \frac{\partial Re}{\partial u_D} = \frac{d}{v} \frac{\partial f}{\partial Re}; \quad f = \begin{cases} \frac{75}{Re}, & \text{when } Re \leq 2300; \\ 0,3164 \cdot Re^{-0,25}, & \text{when } > 2300. \end{cases}$$

By solving the equations system (12.64), we define new values of variables:

$$\{Y\}_{i+1} = \{Y\}_i + \{\Delta Y\}_i \quad (12.66)$$

We shall consider a number of nodes of a joint of a model of a hydraulic system representing a practically important interest for a number of the technical appendices.

Node of joint of mains with leakage of a fluid (fig. 12.16):

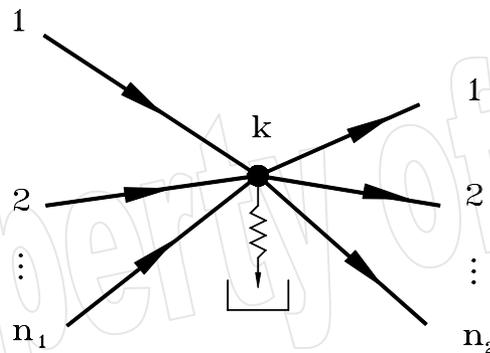


Fig. 12.16. Scheme of a node of joint of mains with leakage of a fluid

The common node of a hydraulic system is considered; into this node come in  $n_1$  and come out  $n_2$  mains. In the common node there are volumetric losses of a fluid, which depend on pressure in the node. The equations system of a motion of a fluid in the common node  $k$  accept the following form:

$$\begin{aligned} \Phi_i = p_{i,k} - p_{i,k-1} + B_i(u_{i,k} - u_{i,k-1}) + \frac{G_i}{8} [f(u_{i,k-1}) + f(u_{i,k})] |u_{i,k-1} + u_{i,k}| \cdot \\ \cdot (u_{i,k-1} + u_{i,k}) + g \sin \alpha = 0, \quad (i = 1, \dots, n_1); \end{aligned} \quad (12.67)$$

$$\begin{aligned} \Phi_j = p_{j,k} - p_{j,k+1} - B_j(u_{j,k} - u_{j,k+1}) - \frac{G_j}{8} [f(u_{j,k}) + f(u_{j,k+1})] |u_{j,k} + u_{j,k+1}| \cdot \\ \cdot (u_{j,k} + u_{j,k+1}) - g \sin \alpha = 0, \quad (j = 1, \dots, n_2); \end{aligned} \quad (12.68)$$

$$\Phi_{n_1+n_2+1} = - \sum_{i=1}^{n_1} A_{i,k} u_{i,k} + \sum_{j=1}^{n_2} A_{j,k} u_{j,k} + a_p p_k = 0; \quad p_{i,k} = p_{j,k} = p_k, \quad (12.69)$$

where:  $A_{i,k}$  – cross-sectional area of  $i$ -th main and  $a_p$  – coefficient of leakages. (12.70)

The nonlinear system of the algebraic equations is solved by the Newton method:

$$[J]_j \{\Delta Y\}_j = -\{\Phi\}_j, \quad (12.71)$$

where  $\{\Delta Y\}_j^T = [\Delta u_{1,k}, \Delta u_{2,k}, \dots, \Delta u_{n_1,k}, \Delta u_{1,k}, \Delta u_{2,k}, \dots, \Delta u_{n_2,k}, \Delta p_k]$

Node of joint of a main with local hydraulic resistances (fig. 12.17):

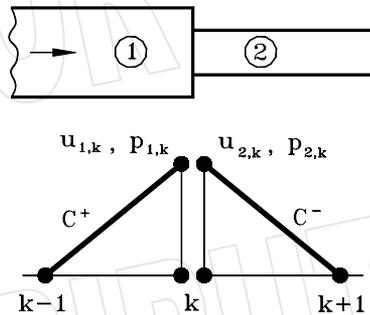


Fig. 12.17. The scheme of a main with local hydraulic losses

As the local hydraulic resistances we will call the resistances to moving of a fluid, which are conditioned by constructive elements of mains and cause a sharp strain of a stream. The equations system of a motion of a fluid in the common node  $k$  accepts the following form:

$$\begin{aligned} \Phi_1 = p_{1,k} - p_{1,k-1} + B(u_{1,k} - u_{1,k-1}) + \frac{G_1}{8} [f(u_{1,k-1}) + f(u_{1,k})] |u_{1,k-1} + u_{1,k}| \cdot \\ \cdot (u_{1,k-1} + u_{1,k}) + g \sin \alpha = 0; \end{aligned} \quad (12.72)$$

$$\begin{aligned} \Phi_2 = p_{2,k} - p_{2,k+1} - B(u_{2,k} - u_{2,k+1}) + \frac{G_2}{8} [f(u_{2,k}) + f(u_{2,k+1})] |u_{2,k} + u_{2,k+1}| \cdot \\ \cdot (u_{2,k} + u_{2,k+1}) - g \sin \alpha = 0; \end{aligned} \quad (12.73 - 12.75)$$

$$\Phi_3 = p_{1,k} - p_{2,k} - \frac{1}{2} \xi_k \rho u_{1,k}^2 \text{ sign } u_{1,k}; \quad A_{1,k} u_{1,k} - A_{2,k} u_{2,k} = 0, \quad (12.76)$$

where  $A_{1,k}$ ,  $A_{2,k}$  – area of the cross-sections of the first and the second main in a node  $k$ ;  $\xi_k$  – coefficient of local resistances, and  $\xi_k = \left( \frac{A_{1,k}}{A_{2,k}} - 1 \right)^2$ . The equations system (12.72) - (12.75) is solved by the

Newton method:

$$[J]_i \{\Delta Y\}_i = -\{\Phi\}_i, \text{ where: } \{\Delta Y\}_i^T = [\Delta p_{1,k}, \Delta p_{2,k}, \Delta u_{1,k}] \quad (12.77)$$

On entry into a main the velocity of a fluid is preset: On entry into a main the velocity of a fluid  $u(t)$  is preset, while the pressure in the considered point of the main is defined by the following relations:

$$p_k = p_{k+1} + B (u(t) - u_{k+1}) + \frac{G}{8} [f(u(t)) + f(u_{k+1})] |u(t) + u_{k+1}| (u(t) + u_{k+1}) + g \sin \alpha \quad (12.78)$$

While the given velocity is defined by the following equation:

$$u(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}, \text{ where } a_i - \text{known coefficients } (i=0 \dots 3)$$

On output of a main the pressure  $p(t)$  is preset: The equation for definition of velocity on output of a main (a node  $k$ ), has the following form:

$$\Phi = p(t) - p_{k-1} + B(u_k - u_{k-1}) + \frac{G}{8} [f(u_{k-1}) + f(u_k)] |u_{k-1} + u_k| (u_{k-1} + u_k) + g \sin \alpha = 0 \quad (12.79)$$

This equation is solved by the Newton method, namely:

$$[J]_i \Delta u_{k,i} = -\Phi_i, \quad (12.80)$$

$$\text{where } [J]_i = \frac{\partial \Phi}{\partial u_k} = B + \frac{G}{8} |u_{k-1} + u_k| \left( 2[f(u_{k-1}) + f(u_k)] + \frac{\partial f}{\partial u_k} (u_{k-1} + u_k) \right) \quad (12.81)$$

The given pressure  $p(t)$  depends on a time as a function of the next form:

$$p(t) = [b_0 + b_1 \sin(b_2 t)] e^{b_3 t}, \text{ where } b_i - \text{known coefficients } (i=0 \dots 3)$$

Node of a main with a hydraulic accumulator (fig. 12.18):

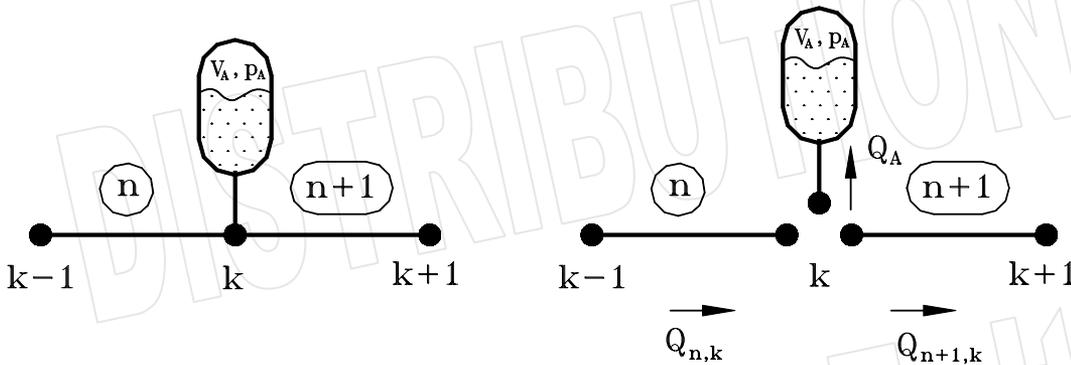


Fig. 12.18. The scheme of a main with a hydraulic accumulator

The balance of expense of a fluid in a node  $k$  of a main is defined by the following relation:

$$Q_{n,k} - Q_{n+1,k} = Q_A, \text{ where } Q_{n,k} = A_{n,k} u_{n,k}; Q_{n+1,k} = A_{n+1,k} u_{n+1,k}; Q_A = \frac{dV_A}{dt} \quad (12.82)$$

where  $A_{n,k}, A_{n+1,k}$  - cross-sectional area of mains  $n$  and  $n+1$  in a node  $k$ ;  $V_A$  - volume of gas in hydraulic accumulator. Then total volume of the accumulator  $V_{AK}$  is described by the following relation:

$$V_{AK} = V_L + V_A, \text{ where } V_L - \text{a volume of a fluid in accumulator} \quad (12.83)$$

By differentiating the expression (12.83) in a time we can determine the expense of a gas in the accumulator, namely:

$$\frac{dV_A}{dt} = -\frac{dV_L}{dt} = -A_A u_A, \quad (12.84)$$

where  $A_A$  – cross-sectional area of a main connected up to accumulator;  $u_A$  – velocity of a fluid coming into accumulator. The dependence between volume  $V_A$  and pressure  $p_A$  in a hydraulic accumulator is defined by the following relation:

$$V_A^\gamma p_A = V_{A_0}^\gamma p_{A_0}, \quad (12.85)$$

where  $V_{A_0}$ ,  $p_{A_0}$  – an initial volume and pressure of a gas in accumulator;  $\gamma$  – isentropic exponent for a gas. By expressing a pressure of gas  $p_A$  from (12.85) and by differentiating it in a time, we obtain the following differential equation:

$$\frac{dp_A}{dt} = \frac{\gamma p_A}{V_A} A_A u_A \quad (12.86)$$

The given differential equation is solved by the Euler method, namely:

$$p_{A,t+\tau} = p_{A,t} + \tau F(V_{A,t}, p_{A,t}), \quad F(V_{A,t}, p_{A,t}) = \frac{\gamma p_{A,t}}{V_{A,t}} A_A u_{A,t}, \quad (12.87)$$

where  $\tau$  – integration step in a time;  $V_{A,t}$ ,  $p_{A,t}$  – volume and pressure of gas in hydraulic accumulator at a moment  $t$ ;  $u_{A,t}$  – velocity of a fluid at a moment  $t$ ;  $p_{A,t+\tau}$  – pressure of gas at a moment  $t+\tau$ . The next guess is made: the pressure in a node  $k$  of a main equals pressure in the hydraulic accumulator, namely:

$$p_{n,k} = p_{n+1,k} = p_A \quad (12.88)$$

Then the equations system written for a node  $k$  at a moment  $t+\tau$ , accepts the following form:

$$\Phi_n = p_{A,t+\tau} - p_{n,k-1,t} + B(u_{n,k,t+\tau} - u_{n,k-1,t}) + \frac{G_n}{8} [f(u_{n,k-1,t}) + f(u_{n,k,t+\tau})] \cdot |u_{n,k-1,t} + u_{n,k,t+\tau}| (u_{n,k-1,t} + u_{n,k,t+\tau}) + g \sin \alpha_n; \quad (12.89)$$

$$\Phi_{n+1} = p_{A,t+\tau} - p_{n+1,k+1,t} - B(u_{n+1,k,t+\tau} - u_{n+1,k+1,t}) - \frac{G_n}{8} [f(u_{n+1,k,t+\tau}) + f(u_{n+1,k+1,t})] \cdot |u_{n+1,k,t+\tau} + u_{n+1,k+1,t}| (u_{n+1,k,t+\tau} + u_{n+1,k+1,t}) - g \sin \alpha_{n+1}.$$

The equations system for definition of velocities  $u_{n,k,t+\tau}$  and  $u_{n+1,k,t+\tau}$  in a node to a main is solved by the Newton method, namely:

$$[J]_i \{\Delta Y\}_i = -\{\Phi\}_i, \quad (12.90)$$

where:  $[J]_i$  – Jacobi matrix after  $i$ -th iteration;

$$[J] = \begin{bmatrix} \frac{\partial \Phi_n}{\partial u_{n,k,t+\tau}} & \frac{\partial \Phi_n}{\partial u_{n+1,k,t+\tau}} \\ \frac{\partial \Phi_{n+1}}{\partial u_{n,k,t+\tau}} & \frac{\partial \Phi_{n+1}}{\partial u_{n+1,k,t+\tau}} \end{bmatrix}; \{\Delta Y\}_i^T = [\Delta u_{n,k,t+\tau} \quad \Delta u_{n+1,k,t+\tau}]; \{\Delta \Phi\}_i^T = [\Phi_n \quad \Phi_{n+1}]. \quad (12.91)$$

The node of a main is connected with a tank (fig. 12.19):

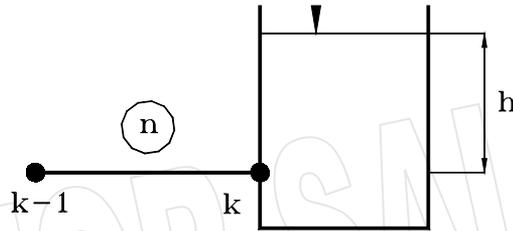


Fig. 12.19. The scheme of a main with a tank

The expense of a fluid coming in the tank, is defined by the next equation:

$$Q_R = \frac{dV}{dt} = A_R \frac{dh}{dt} \quad (12.92)$$

where  $A_R$  – cross-sectional area of the tank;  $h$  – height of a fluid in the tank. Then a balance of expense of a fluid in a node  $k$  of a main is defined as follows (fig. 12.19):

$$A_{n,k} u_{n,k} = Q_R \quad (12.93)$$

By expressing a pressure from expression  $h = \frac{p}{\rho g}$  and by substituting it into (12.92), together with the relation (12.93) we obtain the next differential equation:

$$\frac{dp_{n,k}}{dt} = \rho g \frac{A_{n,k}}{A_R} u_{n,k} \quad (12.94)$$

The given differential equation is solved by the above Euler method, namely:

$$p_{n,k,t+\tau} = p_{n,k,t} + \tau \rho g \frac{A_{n,k}}{A_R} u_{n,k,t} \quad (12.95)$$

Then the equation for determination of velocity of a motion of a fluid in a node  $k$  of a main accepts the following form:

$$\Phi = p_{n,k,t+\tau} - p_{n,k,t} + B(u_{n,k,t+\tau} - u_{n,k-1,t}) + \frac{G}{8} [f(u_{n,k-1,t}) + f(u_{n,k,t+\tau})] \cdot |u_{n,k-1,t} + u_{n,k,t+\tau}| (u_{n,k-1,t} + u_{n,k,t+\tau}) + g \sin \alpha \quad (12.96)$$

The given equation is solved by the above Newton method, namely:

$$[J]_i \Delta u_{n,k,i} = -\Phi_i, \quad (12.97)$$

where

$$[J]_i = \frac{\partial \Phi}{\partial u_{n,k,t+\tau}} = B + \frac{G}{8} |u_{n,k-1,t} + u_{n,k,t+\tau}| \cdot \left( 2[f(u_{n,k-1,t}) + f(u_{n,k,t+\tau})] + \frac{\partial f}{\partial u_{n,k,t+\tau}} (u_{n,k-1,t} + u_{n,k,t+\tau}) \right).$$

### 12.5.2. Input data for the solution of the problem

For the solution of the equations describing a motion of a compressible viscous fluid in a hydraulic system, the *Hydro* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable  $F$ , necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are coded: the number of mains (*parameter nvam*); the number of nodes (*nout*), for which into the file the values of pressure and velocity of a fluid are recorded; the number, indicating through how much

of integration steps the results of calculations (*nstep*) are recorded into the target file. In the *second* line the next parameters are coded: a print code of intermediate results (*kprint*), a density of a fluid (*tankis*), kinematic viscosity (*klampa*), and a velocity of a sound in a hydraulic system (*avel*). If *kprint*=0, the intermediate results are not printed; otherwise, they are printed out.

In the third line, the number of iterations (*niter*), which is necessary for the solution of nonlinear algebraic equations, and a precision of the solution (*toler*) are coded. While in the fourth line, the number of integration steps in a time (*ntime*), a length of a main (*dx*) and also size of an integration step (*dtime*) are coded.

In the subsequent *nvam* lines, the elements of array **Mtop**(*nvam*) are coded. Into each line of the array, the number of a pipeline and also number of its node in which are known values of pressure and velocity of a fluid, are recorded. After of array **Mtop** the elements of **Lout**(*nout*, 2) array are coded line by line. Into each line of the file the line number and elements of **Lout** array are recorded. In the array **Lout** are coded the numbers of mains and numbers of nodes, for which into the file the values of pressure and velocity of a fluid are recorded.

After of array **Lout** into the datafile, the elements of arrays **Plotas**(*nvam*), **Angel**(*nvam*), **P0**(*nvam*) and **V0**(*nvam*) are coded line by line. Into each line of the file the next parameters are coded: the number of a main, cross-sectional area of a main, inclination angle of a main, an initial pressure in a main, and an initial velocity of a fluid in a main. After of arrays **Plotas**, **Angel**, **P0** and **V0** into the datafile, the elements of the **Param**(5, 4) array are coded line by line. Into each line of the datafile are recorded a line number of this array and also four elements of the array **Param**. In addition, the structure of the array **Param**(5, 4) has the following form:

$$\mathbf{Param}(5,4) = \begin{bmatrix} a_0 & a_1 & a_3 & a_4 \\ a_p & 0 & 0 & 0 \\ b_0 & b_1 & b_2 & b_3 \\ p_{A_0} & v_{A_0} & \gamma & 0 \\ A_R & 0 & 0 & 0 \end{bmatrix}$$

In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nvam, nout, nstep, kprint, tankis, klampa, avel, niter, toler, ntime, dx, dtime*  
 Text line \*  
 Arrays **Mtop**(*nvam*)  
 Text line \*  
 Array **Lout**(*nout*, 2)  
 Text line \*  
 Arrays **Plotas**(*nvam*), **Angel**(*nvam*), **P0**(*nvam*), **V0**(*nvam*)  
 Text line \*  
 Array **Param**(5, 4)

**12.5.3. Brief description of the Hydro program solving the problem**

The *Hydro* program was programmed on the *Maple*-language; it consists of the basic program and 18 procedures. All procedures can be divided into three groups: procedures for data entry, for calculation and output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates values of pressures and velocities in each main of a hydraulic system. The

calculation results are output on the monitor and are recorded into files, for that to variables *file\_rezp1* and *file\_rezu1* must be ascribed qualifiers of the target files. Into files *file\_rezp1* and *file\_rezu1* the values of pressure and velocity of a fluid in the indicated nodes of mains are recorded accordingly.

### 12.5.4. An example of use of the Maple-program Hydro

The motion of a fluid in hydraulic system of the next view is considered (fig. 12.20).

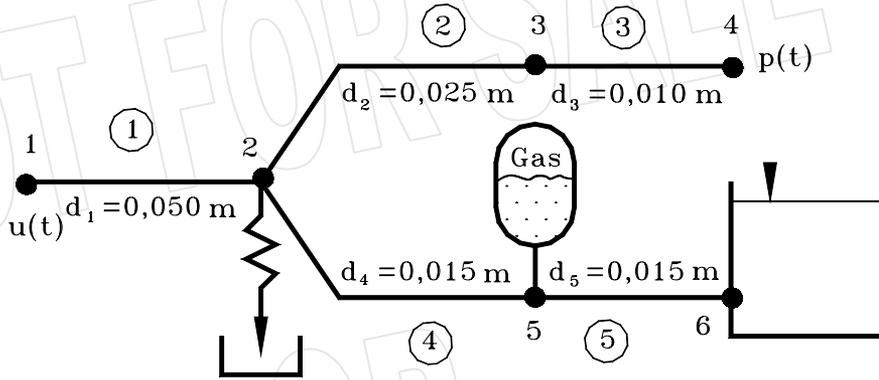


Fig. 12.20. The scheme of the explored hydraulic system

In the node 1 the velocity of a fluid is preset:  $u(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}$ , where  $a_0 = 1,0$  m/s,  $a_1 = 0,2$  m/s,  $a_2 = 2 \cdot \pi \cdot 10^3$  1/s,  $a_3 = 0$ . In the node 2 the volumetric leakages are preset:  $a_p = 10^{-11} \frac{m^5}{N \cdot s}$ . In the node 3 there are local hydraulic losses. In the node 4 the pressure is preset:  $p(t) = [b_0 + b_1 \sin(b_2 t)] e^{b_3 t}$ , where  $b_0 = 4 \cdot 10^5$  Pa,  $b_1 = b_2 = b_3 = 0$ . In the node 5 a hydraulic accumulator with initial volume  $V_{A_0} = 10^{-2} m^3$  and initial pressure  $p_{A_0} = 4 \cdot 10^5$  Pa is linked up. Each main is divided by 10 finite elements.

Input data for the test example:  $nvam=5$ ,  $nout=28$ ,  $nstep=4$ ,  $kprint=1$ ,  $niter=50$ ,  $tankis=10^3$  kg/m<sup>3</sup>,  $klampa=1.004 \times 10^{-6}$  m<sup>2</sup>/s (H<sub>2</sub>O at T=293 K),  $avel=103$  m/s,  $toler=10^{-6}$ ,  $n timer=300$ ,  $dx=0.2$  m,  $dtime=10^{-4}$  s. Initial data are defined as follows:  $u(x, t = 0) = 0$ ,  $p(x, t = 0) = 4 \cdot 10^5$  Pa. On the fig. 12.21 and 12.22 the graphs of change of pressure and velocity in nodes of mains 1-4-5 in a time dependence are represented.

Pressure  $P(x,t)$  Distribution:

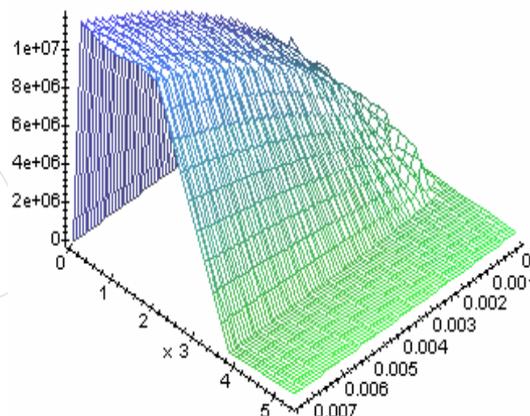


Fig. 12.21. Dependence of pressure of a fluid in a time in nodes of mains 1-4-5

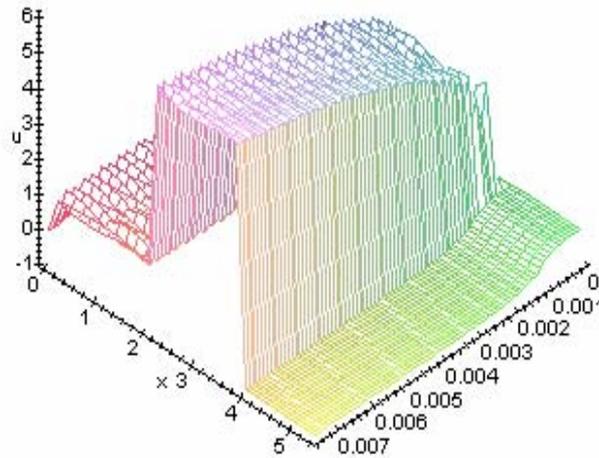
*Velocity  $U(x,t)$  Distribution:*

Fig. 12.22. Dependence of velocity of a fluid in a time in nodes of mains 1-4-5

The *Hydro* program is intended for the solution of equations of motion of a fluid in hydraulic systems respective to the above model. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in PROBLEMS directory of archive attached to the present book in datafiles *Mb9\_5\_new.mws*, *Mb9\_5.dat* and *Mb9\_5\_1.rez* accordingly.

# Chapter 13.

## Applied problems of mechanics - 1

At the solution of dynamic problems of the mechanics of a solid body there is a necessity of definition of moments of inertia of masses of bodies of the composite geometrical forms, and also of systems of bodies relative to various axes of coordinates. As examples the problems over definition of moments of inertia of masses of a body with the composite geometrical form and of systems of such bodies are considered. The package *Maple* allows successfully to obtain analytical expressions of equations of motion and then successfully to solve them, using the numerical methods, and also to reproduce animation of a motion of mechanical systems. As an example the motion of a composite pendulum is considered. In mechanical engineering the mechanical transmissions with a composite motion are widely used, which include the toothed transmissions, clutches etc. As an example of the solution of such problems, the problem of a motion of a mechanical transmission is considered, in which the backlashes in elements of transmission are taken into account.

### 13.1. Calculation of moments of inertia of a solid body

#### 13.1.1. Calculated expressions for definition of moments of inertia of a solid body

At designing of mechanical systems there are problems, for solution of which it is necessary to know allocation of mass in these mechanical systems. The modern mechanical systems include a great many of details, form of which, sometimes, happens very composite. Therefore, for calculation of mechanical systems it is necessary to have a program for calculation of moments of inertia of mass of a body of any composite form. The calculation of *moments of inertia of mass* and other parameters of a body is fulfilled by the numerical method. Most suitable for this problem is the FEM. The explored body is divided by a finite number of three-dimensional quadratic isoparametric elements (fig. 13.1). Use of such finite elements allows precisely to approximate the geometrical form of a body.

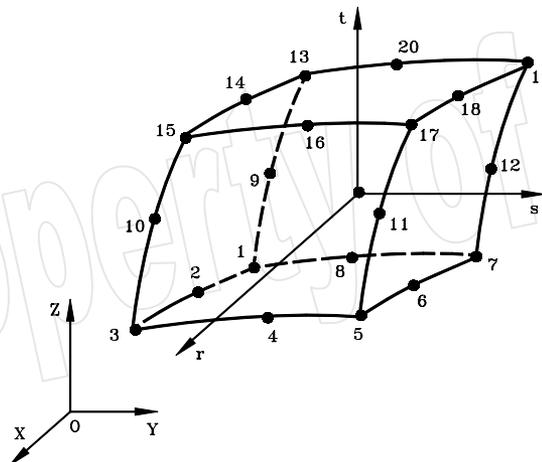


Fig. 13.1. A three-dimensional quadratic isoparametric finite element

The basic characteristics of a body are defined in a rectangular frame  $X - Y - Z$  [106]. The static moments of a solid body are determined by the next relations:

$$S_x = \int_V \rho x \, dV, \quad S_y = \int_V \rho y \, dV, \quad S_z = \int_V \rho z \, dV, \quad \text{where } V - \text{volume of a body.} \quad (13.1)$$

The *moments of inertia* of a solid body relative to coordinate axes are defined as:

$$I_x = \int_V \rho(y^2 + z^2) \, dV; \quad I_y = \int_V \rho(x^2 + z^2) \, dV; \quad I_z = \int_V \rho(x^2 + y^2) \, dV \quad (13.2)$$

The *moments of inertia* of a solid body relative to coordinate planes are defined by the following relations:

$$I_{yoz} = \int_V \rho x^2 \, dV; \quad I_{zox} = \int_V \rho y^2 \, dV; \quad I_{xoy} = \int_V \rho z^2 \, dV \quad (13.3)$$

The *moment of inertia* of a solid body relative to beginning of coordinates is:

$$I_o = \int_V \rho(x^2 + y^2 + z^2) \, dV \quad (13.4)$$

While the *centrifugal moments of inertia* of a solid body are defined by the following relations:

$$I_{yz} = \int_V \rho yz \, dV; \quad I_{zx} = \int_V \rho zx \, dV; \quad I_{xy} = \int_V \rho xy \, dV \quad (13.5)$$

In a finite element *each coordinate* is approximated by the next relation:

$$x = \sum_{i=1}^p N_i(r, s, t) x_i; \quad y = \sum_{i=1}^p N_i(r, s, t) y_i; \quad z = \sum_{i=1}^p N_i(r, s, t) z_i \quad (13.6)$$

where  $N_i(r, s, t)$  - shape functions of a finite element;  $x_i, y_i, z_i$  - coordinates of nodes of a finite element. Three-dimensional integral over volume of a finite element is calculated according to the following formula:

$$I^{(e)} = \int_{V^{(e)}} f(x, y, z) \, dV = \int_{V^{(e)}} f(x, y, z) \det[J] \, dr \, ds \, dt, \quad \text{where } f(x, y, z) - \text{intergrand;} \quad (13.7)$$

$$[J] - \text{determinant of Jacobi matrix and } [J] = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{bmatrix} \quad (13.8)$$

Three-dimensional integral over *volume* of a body is defined by the next relation:

$$I = \sum_{e=1}^{NE} I^{(e)} \tag{13.9}$$

The coordinates of a centre of masses of a body are defined as follows:

$$x_c = \frac{S_x}{V}; \quad y_c = \frac{S_y}{V}; \quad z_c = \frac{S_z}{V}, \text{ where } V - \text{volume of a body} \tag{13.10}$$

The moments of inertia of a solid body relative to central axes  $X', Y', Z'$ , which pass via a point C of a centre of masses and are parallel to coordinate axes  $X, Y, Z$  (fig. 13.2), are determined similarly over the above formulas on the basis of replacement of the coordinates, namely:

$$x' = x - x_c; \quad y' = y - y_c; \quad z' = z - z_c \tag{13.11}$$

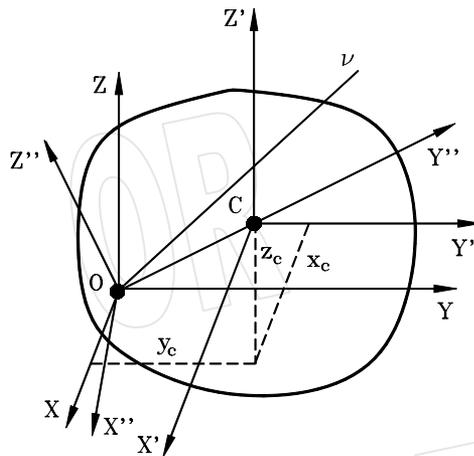


Fig. 13.2. The coordinates systems of a solid body

We draw via a point O in a frame X-Y-Z an axis  $v$ , and turn a solid body around of this axis onto a corner  $\alpha$  counter-clockwise. Then the frame X-Y-Z will occupy a new standing  $X''-Y''-Z''$ . The coordinates of points in the new frame  $X''-Y''-Z''$  are defined by the following relations:

$$\{R''\} = [T(\alpha)]^T \{R\}, \tag{13.12}$$

where  $\{R''\}^T = [x'' \ y'' \ z'']; \{R\}^T = [x \ y \ z]$  and  $[T(\alpha)]$  - matrix of transformation of coordinates,

$$[T(\alpha)] = [E] + [R] \sin \alpha + 2[R]^2 \sin^2 \frac{\alpha}{2}; \tag{13.13}$$

$[E]$  - unit matrix; matrix  $[R]$  of the next view:  $[R] = \begin{bmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{bmatrix}, \{e\}^T = [e_x \ e_y \ e_z]$  - unit

vector in the frame X-Y-Z defining standing of a rotation axis  $Ov$ . The moments of inertia of a solid body in the frame  $X''-Y''-Z''$  are calculated according to the above-mentioned formulas.

### 13.1.2. Input data for the solution of the problem

For solution of a problem concerning of definition of moments of inertia of a solid body the *Mass\_inertia* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable F, necessary qualifier of the file, is ascribed. The data in the

file are placed in the strict order, namely. In the *first* line the number of finite elements (*parameter nelem*), number of nodes (*npoin*) and number of groups of finite elements (*ngr*) are coded. Into the corresponding group of the finite elements are included those elements, which have *identical values* of density of a material.

In the *second* line, a print code of intermediate results (*kprint*), angle of rotation of a frame (*alfa*) and also elements of the **Avector(3)** array are coded. If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. Into the array **Avector(3)** the elements of an unit vector **{e}** are written, which defines a standing of rotation axis **Ov** in the frame **X-Y-Z**.

In each subsequent *nelem* lines, the number of a finite element and the numbers of nodes of this element {array **Mtop(nelem, nnode)**, where *nnode* – number of nodes of the finite element, i.e. *nnode* =20} are coded, and also the number of group, to which belongs this finite element {array **Mgr(nelem)**}. Behind of arrays **Mtop** and **Mgr** into the data file the elements of **Coord(npoin, ndime)** array, where *ndime* – dimensionality of the problem (*ndime*=3), are recorded line by line. Into the *first* column of the **Coord** array the *x*-coordinates, while into the *second* and the *third* columns – the *y*- and *z*-coordinates of nodes of finite elements accordingly are recorded. Into each line of the file the number of a node, and also its (*x, y, z*)-coordinates are recorded.

Behind of the **Coord** array the elements of the **Param(ngr)** array are recorded line by line. Into each line of the datafile, a line number of **Param** array and a value of *density of a material* of a solid body, are recorded. The lines of **Akd** array should correspond to the lines of **Lkd** array. The line number of **Param** array should strictly correspond to the number of group of finite elements. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, npoin, kprint, alfa, Avector(3)*  
 Text line \*  
 Arrays **Mtop(nelem, nnode), Mgr(nelem)**  
 Text line \*  
 Array **Coord(npoin, ndime)**  
 Text line \*  
 Array **Param(ngr)**

### 13.1.3. Brief description of the Mass\_inertia program solving the problem

The *Mass\_inertia* program was programmed on the Maple-language; it consists of the basic program and 13 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The program calculates *coordinates of centre of masses* and *moments of inertia* of a solid body of the arbitrary form, which consists of the *various materials*. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a qualifier of a target file. Into the file *file\_rez1* values of coordinates of centre of masses and moments of inertia of a solid body are recorded.

### 13.1.4. An example of use of the Maple-program Mass\_inertia

As an example of application of the *Mass\_inertia* program, the definition of moments of inertia of a solid body represented on the *fig. 13.3*, is considered. The explored solid body consists of two

groups of finite elements: to the *first* group concern the *first* and the *second* finite element, while to the *second* group – the *third* finite element.

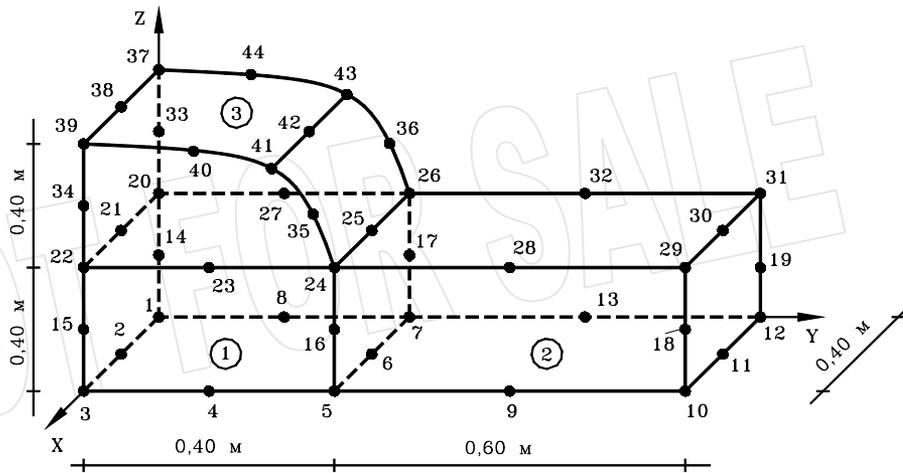


Fig. 13.3. The calculated scheme of the explored solid body

Input data for the test example: number of the finite elements  $nelem=3$ , number of nodes  $npoin=44$ , number of groups of finite elements  $ngr=2$ , number of boundary conditions  $nbond=1$ ,  $kprint=0$ ,  $nq=4$ ,  $nst=2$ ,  $nforc=1$ ,  $ksolve=0$ ,  $niter=500$ ,  $toler=10^{(-9)}$ ,  $ngama=1$ ,  $nkd=1$ ,  $ntime=500$ ,  $dtime=10^{(-4)}$  s,  $neigen=5$ ,  $nitero=100$ ,  $nout=2$ ,  $nstep=2$ ,  $tolero=10^{(-6)}$ .

Density of a material of the first group  $\rho_1 = 2700 \frac{kg}{m^3}$  and the second group  $\rho_2 = 7850 \frac{kg}{m^3}$  accordingly; angle of rotation of the frame

$\alpha = 45^\circ$ ; unit vector  $\{e\}^T = [0 \ 0,707 \ 0,707]$ . The remaining data necessary for calculation, are represented on the fig. 13.3.

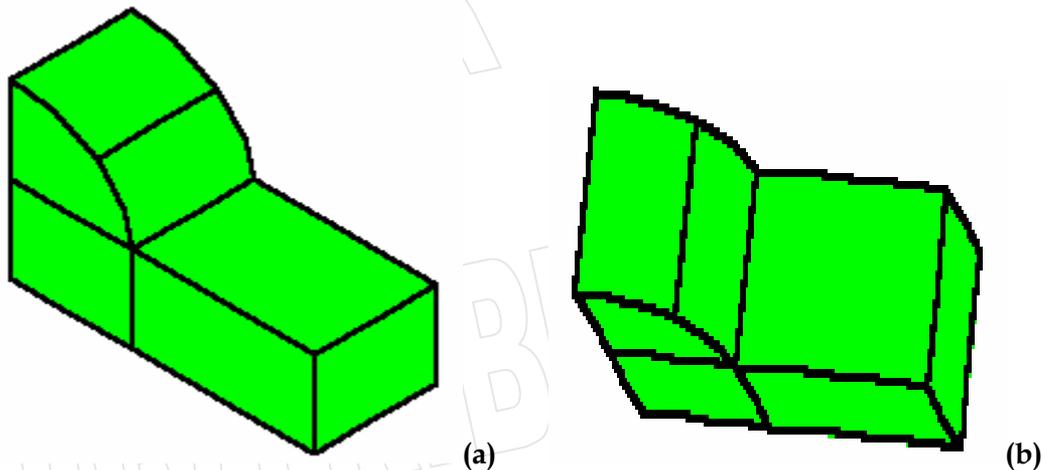
Results of calculation over the test example:

Moments of Inertia, Mass, Volume, Coordinates of Centre Mass:

- Inertia tensor  $I_{xx}=3.142988e+02$
- Inertia tensor  $I_{xy}=-5.648783e+01$
- Inertia tensor  $I_{xz}=-6.198557e+01$
- Inertia tensor  $I_{yy}=1.986480e+02$
- Inertia tensor  $I_{yz}=-7.965422e+01$
- Inertia tensor  $I_{zz}=2.036212e+02$
- Mass of body  $mass=8.247217e+02$
- Volume of body  $V=2.100282e-01$
- Coordinate of centre of mass  $xc=2e-01$
- Coordinate of centre of mass  $yc=3.424660e-01$
- Coordinate of centre of mass  $zc=3.757968e-01$
- Inertia tensor  $I_{xcxc}=1.011031e+02$
- Inertia tensor  $I_{xcyc}=1.135817e-14$
- Inertia tensor  $I_{xczc}=2.365034e-15$
- Inertia tensor  $I_{ycyc}=4.918921e+01$
- Inertia tensor  $I_{yczc}=2.648553e+01$
- Inertia tensor  $I_{zczc}=7.390651e+01$
- Rotation about an Axes:

Inertia tensor  $I_{xx\_new}=2.936519e+02$   
 Inertia tensor  $I_{xy\_new}=-2.880000e+01$   
 Inertia tensor  $I_{xz\_new}=-5.249549e+01$   
 Inertia tensor  $I_{yy\_new}=1.504609e+02$   
 Inertia tensor  $I_{yz\_new}=-8.997157e+01$   
 Inertia tensor  $I_{zz\_new}=2.724303e+02$   
 Coordinate of centre of mass  $x_{c\_new}=1.581019e-01$   
 Coordinate of centre of mass  $y_{c\_new}=4.473306e-01$   
 Coordinate of centre of mass  $z_{c\_new}=2.709322e-01$   
 Inertia tensor  $I_{xcxc\_new}=6.808895e+01$   
 Inertia tensor  $I_{xcyc\_new}=2.952597e+01$   
 Inertia tensor  $I_{xczc\_new}=-1.716951e+01$   
 Inertia tensor  $I_{ycyc\_new}=6.931003e+01$   
 Inertia tensor  $I_{yczc\_new}=9.979061e+00$   
 Inertia tensor  $I_{zczc\_new}=8.678953e+01$

Initial standing of a solid body and its standing at turning around of the given axis are represented on the *fig. 13.4*.



*Fig. 13.4. Standings of the explored solid body: a – initial; b – at rotation around the given axis*

The *Mass\_inertia* program is intended for calculation of coordinates of centre of masses and moments of inertia of a solid body. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles *Mb10\_1\_new.mws*, *Mb10\_1.dat* and *Mb10\_1\_1.rez* accordingly.

### 13.2. Calculation of inertia moments of a system of solid bodies

The dynamic properties of mechanical systems depend on many parameters. The *moments of inertia* of solid bodies, and also *moments of inertia of all mechanical system* as a whole concern to these parameters. The composite mechanical systems consist of separate bodies of the manifold form and from various materials, densities of which are various. As examples of such mechanical systems can be: automobile and railway trains, robots, lifting-transport machines, building-road machines, metal-working machines etc. For the solution of dynamic problems of such mechanical systems with concentrated parameters it is necessary to determine moments of inertia of a system of solid bodies.

### 13.2.1. The calculated expressions for determination of inertia moments of a system of solid bodies

For determination of *inertia moments* of a system of solid bodies the common frame **X-Y-Z** is introduced. Each solid body has own frame **X<sub>i</sub>-Y<sub>i</sub>-Z<sub>i</sub>**, which is located at a centre of masses of this body (fig. 13.5).

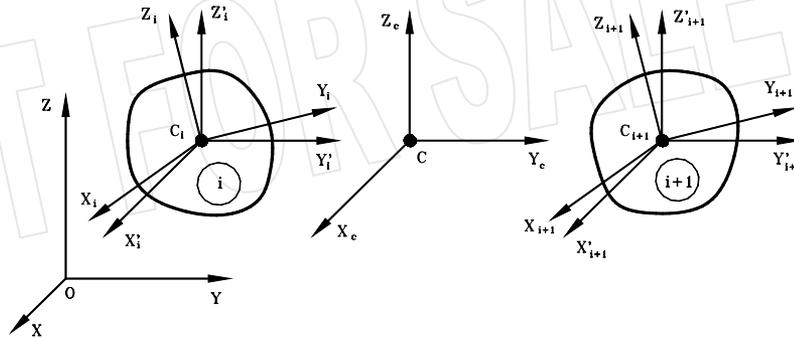


Fig. 13.5. Frames

The *orientation* of central coordinate axes **X<sub>i</sub>**, **Y<sub>i</sub>** and **Z<sub>i</sub>** of the solid bodies relative to common frame **X-Y-Z** is defined by the following dependence:

$$\begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} = [T] \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}, \text{ where } [T] - \text{matrix of transformation of coordinates} \quad (13.14)$$

The *coordinates of centre of masses* of a system of solid bodies are defined by the following relations:

$$x_c = \frac{\sum_{i=1}^n m_i x_i}{\sum_{i=1}^n m_i}; \quad y_c = \frac{\sum_{i=1}^n m_i y_i}{\sum_{i=1}^n m_i}; \quad z_c = \frac{\sum_{i=1}^n m_i z_i}{\sum_{i=1}^n m_i} \quad (13.15)$$

Moments of inertia of a solid body relative to axes **X'**, **Y'** and **Z'**, which pass via a centre of masses and are parallel to axes **X**, **Y** and **Z**, are defined by the following relations:

$$\begin{aligned} I_{y'_i} &= (t_{11}^2 + t_{31}^2) I_{y_{i0}z_i} + (t_{12}^2 + t_{32}^2) I_{z_{i0}x_i} + (t_{13}^2 + t_{33}^2) I_{x_{i0}y_i} + \\ &+ 2(t_{11}t_{12} + t_{31}t_{32}) I_{x_i y_i} + 2(t_{11}t_{13} + t_{31}t_{33}) I_{z_i x_i} + 2(t_{12}t_{13} + t_{32}t_{33}) I_{y_i z_i}; \\ I_{x'_i} &= (t_{21}^2 + t_{31}^2) I_{y_{i0}z_i} + (t_{22}^2 + t_{32}^2) I_{x_{i0}z_i} + (t_{23}^2 + t_{33}^2) I_{x_{i0}y_i} + \\ &+ 2(t_{21}t_{22} + t_{31}t_{32}) I_{x_i y_i} + 2(t_{21}t_{23} + t_{31}t_{33}) I_{x_i z_i} + 2(t_{22}t_{23} + t_{32}t_{33}) I_{y_i z_i}; \\ I_{z'_i} &= (t_{11}^2 + t_{21}^2) I_{y_{i0}z_i} + (t_{12}^2 + t_{22}^2) I_{x_{i0}z_i} + (t_{13}^2 + t_{23}^2) I_{x_{i0}y_i} + \\ &+ 2(t_{11}t_{12} + t_{21}t_{22}) I_{x_i y_i} + 2(t_{11}t_{13} + t_{21}t_{23}) I_{x_i z_i} + 2(t_{12}t_{13} + t_{22}t_{23}) I_{y_i z_i}, \end{aligned} \quad (13.16)$$

where  $t_{ij}$  – elements of a matrix of transformation of coordinates. Centrifugal moments of inertia of a solid body relative to axes  $X'$ ,  $Y'$  and  $Z'$ , which pass via a centre of masses and are parallel to axes  $X$ ,  $Y$  and  $Z$ , are defined by the following relations:

$$\begin{aligned}
 I_{x'y'} &= t_{11}t_{21}(I_z - I_x) + t_{12}t_{22}(I_z - I_y) + (t_{11}t_{22} + t_{12}t_{21}) I_{xy} + \\
 &\quad + (t_{11}t_{23} + t_{13}t_{21}) I_{xz} + (t_{12}t_{23} + t_{13}t_{22}) I_{yz}; \\
 I_{z'x'} &= t_{11}t_{31}(I_z - I_x) + t_{12}t_{32}(I_z - I_y) + (t_{11}t_{32} + t_{12}t_{31}) I_{xy} + \\
 &\quad + (t_{11}t_{33} + t_{13}t_{31}) I_{xz} + (t_{12}t_{33} + t_{13}t_{32}) I_{yz}; \\
 I_{y'z'} &= t_{21}t_{31}(I_z - I_x) + t_{22}t_{32}(I_z - I_y) + (t_{21}t_{32} + t_{22}t_{31}) I_{xy} + \\
 &\quad + (t_{22}t_{33} + t_{23}t_{32}) I_{yz} + (t_{21}t_{33} + t_{33}t_{31}) I_{zx}.
 \end{aligned}
 \tag{13.17}$$

The *axial* and *centrifugal inertia moments* of a system of solid bodies relative to axes  $X_c$ ,  $Y_c$  and  $Z_c$ , passing via centre of masses of the system, are determined as follows:

$$\begin{aligned}
 I_{x_c} &= \sum_{i=1}^n I_{x'_i} + (x_i - x_c)^2 m_i; \quad I_{y_c} = \sum_{i=1}^n I_{y'_i} + (y_i - y_c)^2 m_i; \\
 I_{z_c} &= \sum_{i=1}^n I_{z'_i} + (z_i - z_c)^2 m_i; \quad I_{x_c y_c} = \sum_{i=1}^n I_{x'_i y'_i} + (x_i - x_c)(y_i - y_c) m_i; \\
 I_{y_c z_c} &= \sum_{i=1}^n I_{y'_i z'_i} + (y_i - y_c)(z_i - z_c) m_i; \quad I_{z_c x_c} = \sum_{i=1}^n I_{z'_i x'_i} + (z_i - z_c)(x_i - x_c) m_i
 \end{aligned}
 \tag{13.18}$$

### 13.2.2. Input data for the solution of the problem

For solution of a problem concerning of definition of moments of inertia of a system of solid body the *Mass\_system* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable *F*, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely.

In the *first* line the number of solid bodies (*parameter nelem*), number of lines which link some solid bodies (*nline*) and a print code of intermediate results (*kprint*) are coded. If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out.

In each subsequent *nelem* lines, the number of a solid body, its mass, moments of inertia relative to planes  $I_{x_0 y_0}$ ,  $I_{y_0 z_0}$ ,  $I_{z_0 x_0}$ , and centrifugal moments of inertia  $I_{x_0 y_0}$ ,  $I_{y_0 z_0}$ ,  $I_{z_0 x_0}$  (array *Amase(nelem, 7)*). Behind of array *Amase(nelem, 7)* and into the data file the elements of *Coord(nelem, ndime)* array, where *ndime* – dimensionality of the problem (*ndime*=3), are recorded line by line. Into the *first* column of the *Coord* array the *x*-coordinates, while into the *second* – *y*-coordinates of nodes, and into the *third* – *z*-coordinates of centres of masses of solid bodies are recorded. Into each line of the file the number of a solid body, and also its (*x,y,z*)-coordinates of centre of masses are recorded.

Behind of the *Coord* array the elements of the *Tensor(3\*nelem, nmode)* array are recorded line by line. Into the array *Tensor* the matrixes of transformation of coordinates **[T]** of each solid body are

recorded one after another. Into each line of the datafile, the number of a solid body and three values of direction cosines of the matrix  $[T]$  are recorded.

Behind of the **Tensor** array into the datafile the elements of the arrays **Mline**(*nline*, 2) and **Cline**(*nline*, 6) are recorded line by line. Into columns of the array **Mline** the numbers of two solid bodies, between which the straight line is traced, are recorded. Into columns of the array **Cline** the coordinates  $x_i$ ,  $y_i$  and  $z_i$  of centre of masses of the *first* and the *second* solid bodies, between which the straight line is traced, are recorded. Into each line of the datafile are recorded the parameters such as: a line number, two numbers of solid bodies and the coordinates  $x_i$ ,  $y_i$ ,  $z_i$  of the *first* and the *second* solid bodies. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nelem, nline, kprint*  
Text line \*  
Array *Amase(nelem, 7)*  
Text line \*  
Array *Coord(nelem, ndime)*  
Text line \*  
Array *Tensor(3\*nelem, ndime)*  
Text line \*  
Arrays *Mline(nline, 2), Cline(nline, 6)*

### 13.2.3. Brief description of the Mass\_system program solving the problem

The *Mass\_system* program was programmed on the *Maple*-language; it consists of the basic program and 10 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of finite elements, and number of nodes. The given program calculates *coordinates of centre of masses* and *moments of inertia* of a system of solid bodies. The calculation results are output on the monitor and are recorded into a file, for that to variable *file\_rez1* of the program must be ascribed a *qualifier* of a target file. Into the file *file\_rez1* values of *coordinates of centre of masses* and *moments of inertia* of a system of solid bodies are recorded.

### 13.2.4. An example of use of the Maple-program Mass\_system

As an example of application of the *Mass\_system* program, the definition of inertia moments of the system of solid bodies, represented on the *fig. 13.6*, is considered.

Input data for the test example: number of the solid bodies *nelem*=10, the number of straight lines which link some solid bodies *nline*=8, the masses of solid bodies:  $m_1 = m_3 = m_7 = m_9 = 20 \text{ kg}$ ;  $m_2 = 100 \text{ kg}$ ;  $m_4 = 200 \text{ kg}$ ;  $m_5 = 50 \text{ kg}$ ;  $m_6 = 500 \text{ kg}$ ;  $m_8 = 150 \text{ kg}$ ;  $m_{10} = 70 \text{ kg}$ . The moments of inertia relative to centre of masses of a solid body:  $I_1 = I_3 = I_7 = I_9 = 0,60 \text{ kg} \cdot \text{m}^2$ ;  $I_2 = 15,0 \text{ kg} \cdot \text{m}^2$ ;  $I_4 = 30,0 \text{ kg} \cdot \text{m}^2$ ;  $I_5 = 3,0 \text{ kg} \cdot \text{m}^2$ ;  $I_6 = 125,0 \text{ kg} \cdot \text{m}^2$ ;  $I_8 = 20,0 \text{ kg} \cdot \text{m}^2$ ;  $I_{10} = 5,0 \text{ kg} \cdot \text{m}^2$ . The remaining data necessary for calculation, are represented on the *fig. 13.6*.

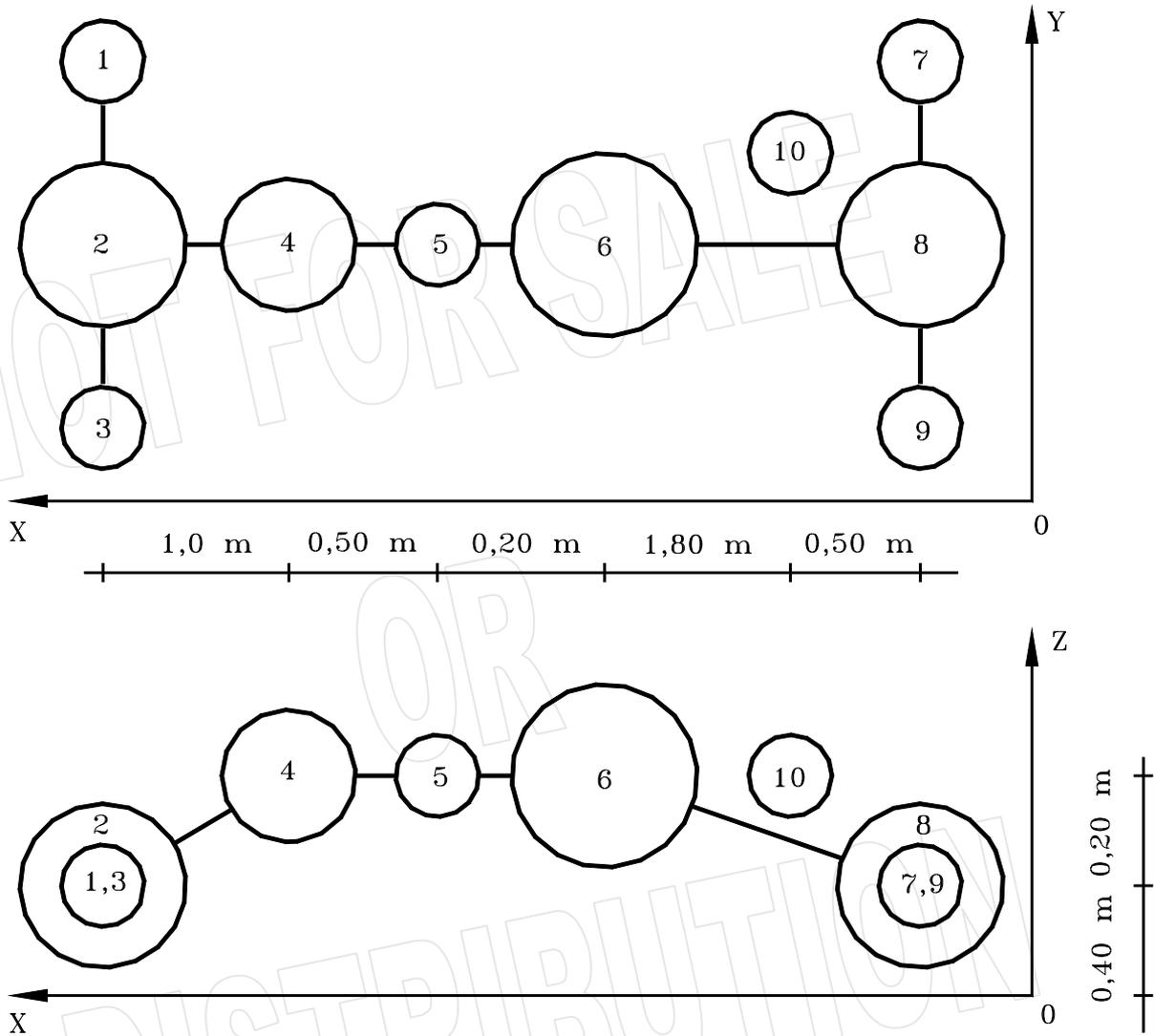


Fig. 13.6. System of solid bodies (form of each solid body is a sphere)

**Results of calculation over the test example:**

**Mass, Coordinates of Center Mass and Inertia Moments:**

Mass of system of rigid bodies  $mass=1.15e+03$   
 Coordinate of center of mass  $x_c=2.147826e+00$   
 Coordinate of center of mass  $y_c=1.030434e+00$   
 Coordinate of center of mass  $z_c=5.426086e-01$   
 Inertia moments about yoz plane  $I_{yoz}=1.910269e+03$   
 Inertia moments about zox plane  $I_{zox}=2.968347e+02$   
 Inertia moments about xoy plane  $I_{xoy}=2.098121e+02$   
 Inertia moments  $I_{xy}=-5.767391e+01$   
 Inertia moments  $I_{yz}=2.008695e+00$   
 Inertia moments  $I_{zx}=2.975652e+01$

The scheme of the system of solid bodies is represented on the following fig. 13.7.

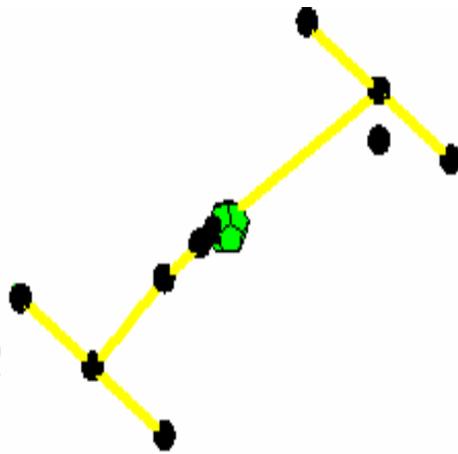


Fig. 13.7. The scheme of the system of solid bodies

The *Mass\_system* program is intended for calculation of coordinates of centre of masses and moments of inertia of a system of solid bodies. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb10\_2\_new.mws*, *Mb10\_2.dat* and *Mb10\_2\_1.rez* accordingly.

### 13.3. Nonlinear oscillations of mechanical systems

In some mechanical systems the connection between separate solid bodies can be nonlinear. At investigation of oscillations of such mechanical systems there are difficulties at definition of amplitude-frequency characteristics. Therefore, such problems have practical significance at investigation of nonlinear oscillations of mechanical systems.

#### 13.3.1. The calculated expressions for description of nonlinear oscillations of mechanical systems

A nonlinear mechanical system consisting of solid bodies, which are connected by nonlinear elastic and dissipative elements, is considered (fig. 13.8).

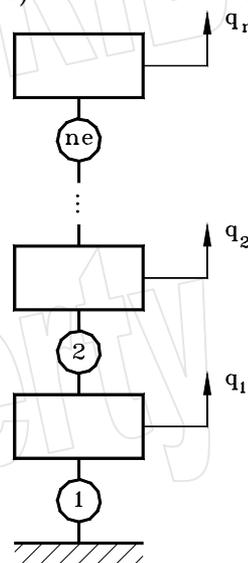


Fig. 13.8. The scheme of the mechanical system (by circle are marked the elements, consisting of a spring and a damper)

The *equations of motion* of a mechanical system are defined on the basis of the Lagrange equation of the second type, namely:

$$\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{q}_i} \right) - \frac{\partial T}{\partial q_i} + \frac{\partial \Phi}{\partial \dot{q}_i} + \frac{\partial \Pi}{\partial q_i} = Q_i \quad (13.19)$$

where:  $T$  - kinetic energy of a mechanical system:  $T = \sum_{i=1}^n \frac{1}{2} m_i \dot{q}_i^2$  (13.20)

$$\Pi$$
 - potential energy of a mechanical system:  $\Pi = \sum_{i=1}^{ne} \frac{1}{2} k_i (\Delta_i)^2$  (13.21)

$$\Delta_i = q_i - q_{i-1} \quad (13.22)$$

$$\Phi$$
 - dissipation function:  $\Phi = \sum_{i=1}^{ne} \frac{1}{2} h_i (\dot{\Delta}_i)^2$  (13.23)

$$\dot{\Delta}_i = \dot{q}_i - \dot{q}_{i-1} \quad (13.24)$$

$q_i, \dot{q}_i$  - the generalized coordinates and velocity;  $Q_i$  - the generalized force;  $t$  - time;  $m_i$  - mass of a body;  $k_i, h_i$  - coefficients of stiffness and damping;  $n$  - the number of bodies;  $ne$  - the number of elements. The generalized force is represented as follows:

$$Q_i = A_{i,0} + \sum_{k=1}^{nf} A_{i,k} \cos kvt + B_{i,k} \sin kvt \quad (13.25)$$

where  $A_{i,k}, B_{i,k}$  - known coefficients;  $v$  - excitation frequency;  $nf$  - harmonics of excitation. The coefficients of *stiffness* and *damping* are functions of the generalized coordinates, namely:

$$k_i = \sum_{j=0}^{nk} k_{ij} \Delta^j; \quad h_i = \sum_{j=0}^{nh} h_{ij} \Delta^j; \quad (i=1 \dots ne). \quad (13.26)$$

The equations system of motion of a mechanical system have the next form:

$$[M]\{\ddot{q}\} + \{F(q, \dot{q})\} = \{Q\}. \quad (13.27)$$

The generalized coordinate  $q_i$  with a period  $T$  is expanded into Fourier series:

$$q_i(t) = \frac{1}{2} a_{i,0} + \sum_{k=1}^{nq} a_{i,k} \cos k\omega t + b_{i,k} \sin k\omega t; \quad \omega = \frac{2\pi}{T}; \quad (i=1..n) \quad (13.28)$$

By substituting expressions (13.28) of the generalized coordinates into the equations system of a motion (13.27), we shall receive a vector of the equations with unknown coefficients  $a_{i,k}$  and  $b_{i,k}$ , namely:

$$\{\Phi(a,b)\} = 0 \quad (13.29)$$

Each  $i$ -th equation, as a real periodic function with a period  $T$ , is expanded into Fourier series of the following form:

$$\Phi_i(t) = \frac{1}{2} z_{i,0} + \sum_{k=1}^{nq} (z_{c_{i,k}} \cos k\omega t + z_{s_{i,k}} \sin k\omega t), \text{ where } z_{i,0} = \frac{2}{T} \int_{-T/2}^{T/2} \Phi_i dt, \quad (13.30)$$

$$z_{c_{i,k}} = \frac{2}{T} \int_{-T/2}^{T/2} \Phi_i \cos k\omega t dt, \quad z_{s_{i,k}} = \frac{2}{T} \int_{-T/2}^{T/2} \Phi_i \sin k\omega t dt \quad (13.31-13.33)$$

Then, the equations system of a motion can be presented as follows:

$$\{z(a,b)\} = [z_1, \dots, z_n]^T = 0, \quad \{z_i\}^T = [z_{i,0} \ z_{c_{i,1}} \ z_{s_{i,1}} \ \dots \ z_{c_{i,nq}} \ z_{s_{i,nq}}] \quad (13.34)$$

The obtained equations system (13.34) is a system of nonlinear algebraic equations relative to coefficients  $a_{i,k}$  and  $b_{i,k}$ . The given equations system is solved by the Newton method, namely:

$$[J]_i \{\Delta X\}_i = -\{Z\}_i, \text{ where: } [J]_i - \text{Jacobi matrix of the form } [J]_i = \frac{\partial \{Z\}_i}{\partial \{X\}}; \quad (13.35)$$

$$\{X\} - \text{vector of unknowns, } \{X\}^T = \{a_{1,0} \ a_{1,1} \ b_{1,1} \ a_{1,2} \ b_{1,2} \ \dots \ a_{n,nq} \ b_{n,nq}\}. \quad (13.36)$$

### 13.3.2. Input data for the solution of the problem

For solution of a problem concerning of definition of coefficients of the corresponding harmonics of variables of a nonlinear mechanical system the *Amplitude* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable  $F$ , necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are recorded: number of bodies (*parameter nbody*); number of elastic and dissipative elements (*nelem*); number of harmonics (*nq*); number of addends defining coefficients of a stiffness (*nk*); number of addends defining coefficients of a damping (*nh*); number of working external forces (*nforce*) and the number of harmonics of external forces (*nf*).

In the *second* line the next parameters are recorded: a print code of intermediate results (*kprint*); number of steps on frequency (*nw*); a minimum frequency (*wmin*); a step of frequency (*dw*); number of iterations (*niter*) and precision of the solution (*toler*) at the solution of a system of the nonlinear equations by the Newton method. If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out.

In each subsequent *nelem* lines, the elements of the array **Amase(nbody)** are coded. Into each line of the datafile the number of a solid body and its mass are recorded. Behind of the array **Amase(nbody)** into the datafile the elements of the **Mtop(nelem, 2)** array are recorded line by line. Into this array are recorded the numbers of bodies, between which elements of elasticity and

damping are located. Into each line of the datafile the number of an element and also numbers of bodies are recorded. Behind of the array **Mtop** into the datafile the elements of the **Ak**(*nelem, nk*) and **Ah**(*nelem, nh*) arrays are recorded line by line. Into arrays **Ak** and **Ah** the addends for determination of coefficients of stiffness and dampings are recorded. Into each line of the datafile the number of an element and also *nk* coefficients of stiffness and *nh* coefficients of dampings are recorded.

Behind of the arrays **Ak** and **Ah** into the datafile the elements of the arrays **Mjega**(*nforce*), **Pw**(*nforce*) and **Pjega**(*nforce*) are recorded line by line. Into each line of the array **Mjega** the number of a body onto which the external force affects is recorded; into the array **Pw** the frequency of excitations is recorded, at last into the array **Pjega** the components of external force are recorded. Into each line of the datafile the number of a body and *nf* of components of external forces are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nbody, nelem, nq, nk, nh, nforce, nf*  
 Text line \*  
*kprint, nw, wmin, dw, niter, toler*  
 Text line \*  
 Array *Amase(nbody)*  
 Text line \*  
 Array *Mtop(nelem, 2)*  
 Text line \*  
 Arrays *Ak(nelem, nk), Ah(nelem, nh)*  
 Text line \*  
 Arrays *Mjega(nforce), Pw(nforce), Pjega(nforce)*

### 13.3.3. Brief description of the Amplitude program solving the problem

The *Amplitude* program was programmed on the *Maple*-language; it consists of the basic program and 17 procedures. All procedures can be divided into three groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of bodies, the number of steps on frequency and precision of solution. The program calculates coefficients of the corresponding harmonics of the variables of a nonlinear mechanical system, and also eigenvalues and own frequencies of a linear mechanical system. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1* and *file\_rez2* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of coefficients of the corresponding harmonics of the variables of a nonlinear mechanical system, while into the file *file\_rez2* eigenvalues and own frequencies of a linear mechanical system are recorded accordingly.

### 13.3.4. An example of use of the Maple-program Amplitude

As an example of application of the *Amplitude* program, the oscillations of the mechanical system, represented on the *fig. 13.9*, for definition of coefficients of Fourier series of variables of the given system are considered.

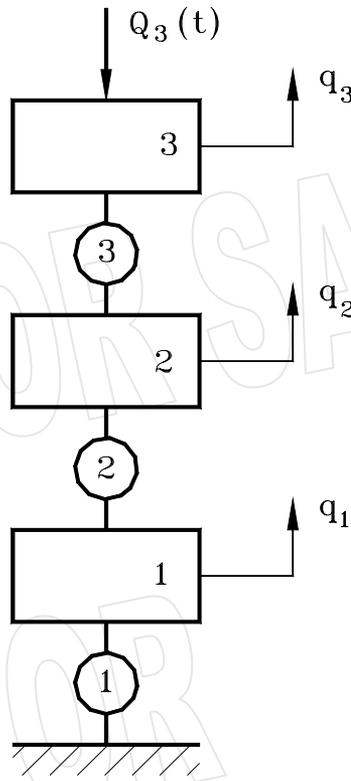


Fig. 13.9. The scheme of explored system

Input data for the test example: number of the solid bodies  $nbody=3$ ; the number of elastic bodies  $nelem=3$ ; number of harmonics  $nq=3$ ; number of addends of coefficients of stiffness  $nk=2$  and dampings  $nh=2$ ; number of active external forces  $nforce=1$ ; number of components of external force  $nf=5$ ;  $kprint=0$ ,  $nw=21$ ,  $wmin=5$  Hz,  $dw=1$  Hz,  $niter=5$ ,  $toler=10^{(-6)}$ . The masses of bodies:  $m_1 = 10$  kg ;  $m_2 = 5$  kg ;  $m_3 = 2$  kg . Coefficients of stiffness of elements:  $k_{1,0} = 5 \cdot 10^5$  N/m ;  $k_{1,1} = 0$  ;  $k_{2,0} = 2 \cdot 10^5$  N/m ;  $k_{2,1} = 0$  ;  $k_{3,0} = 1 \cdot 10^5$  N/m ;  $k_{3,1} = 10^7$  N/m. Coefficients of dampings of elements:  $h_{1,0} = 5 \cdot 10^{-2}$  Ns/m ;  $h_{3,0} = 10^{-2}$  Ns/m ;  $h_{2,0} = 3 \cdot 10^{-2}$  Ns/m ;  $h_{1,1} = h_{2,1} = h_{3,1} = 0$ . Excitation frequency  $\nu = 15$  Hz and external force  $Q_{31} = 5$  N ;  $Q_{32} = 2$  N ;  $Q_{33} = 6$  N ;  $Q_{34} = 10$  N .

Results of calculation over the test example:

*Eigenvalues (real and imaginary part) and own frequencies:*

Re[1]= -6.386769e-03	Im[1]= 3.210184e+02	frequency[1]= 1.791698e+01 Hz
Re[2]= -6.386769e-03	Im[2]= -3.210184e+02	frequency[2]= 1.791698e+01 Hz
Re[3]= -3.132243e-03	Im[3]= 2.470895e+02	frequency[3]= 1.571908e+01 Hz
Re[4]= -3.132243e-03	Im[4]= -2.470895e+02	frequency[4]= 1.571908e+01 Hz
Re[5]= -9.809867e-04	Im[5]= 1.260711e+02	frequency[5]= 1.122814e+01 Hz
Re[6]= -9.809867e-04	Im[6]= -1.260711e+02	frequency[6]= 1.122814e+01 Hz

On the fig. 13.10 the dependences of coefficients of the corresponding harmonics of the third variable of the nonlinear mechanical system from frequency are represented.

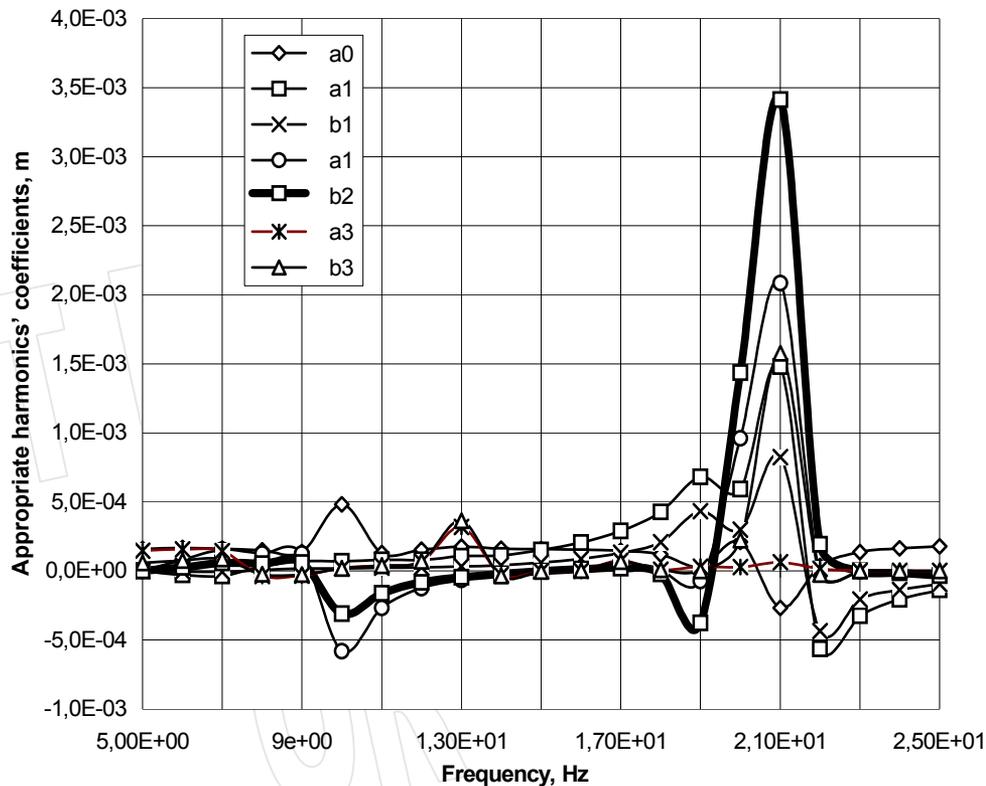


Fig. 13.10. Dependences of coefficients of the corresponding harmonics of the third variable of the nonlinear mechanical system from frequency

The *Amplitude* program is intended for the solution of problem on determination of coefficients of the corresponding harmonics of variables of a nonlinear mechanical system. The source module of the program in Maple-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files Mb10\_3\_new.mws, Mb10\_3.dat and Mb10\_3\_1.rez accordingly.

### 13.4. Derivation and solution of equations of motion of a mechanical system with the concentrated parameters

The Maple package gives an opportunity to output and to solve equations of motion of mechanical systems with the concentrated parameters. At a composite motion of a mechanical system the equations of motion of this system have an unwieldy view and at their deduction the errors of a various kind are possible. The equations of motion can successfully be deduced, using the Lagrange equation of the *second* type. As an example of a mechanical system with the concentrated parameters, the system of an interesting construction consisting of a central pendulum and four satellites is explored. For providing of an unstable motion of such mechanical system the stationary magnets are used.

#### 13.4.1. The calculated expressions for deduction of equations of motion of a mechanical system

The motion of a mechanical system in a vertical plane is considered. A mechanical system consists of six masses  $\mathbf{m}_i$  ( $i=1 \dots 6$ ). The centre of masses of each body on the *fig. 13.11* is indicated by points  $\mathbf{c}_i$  ( $i=1 \dots 6$ ). Thickness of a constant magnet is designated  $\Delta_i$  ( $i=1 \dots 3$ ), and the distances up to

magnets from points  $O$ ,  $O_1$  and  $O_2$  are designated as  $h_1$ ,  $h_2$  and  $h_3$  accordingly. In each body, excepting the second, the stationary magnets are located. A poles of stationary magnets in points  $O_4$ ,  $O_5$  and  $O_6$  coincide with poles of stationary magnets located in the rotation bodies. Thus, onto the body which is located near to a constant magnet, a magnetic force affects. The equations of motion of a mechanical system are deduced, using the Lagrange equation of the second type, namely:

$$\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{\mathbf{q}}} \right) - \frac{\partial T}{\partial \mathbf{q}} + \frac{\partial \Phi}{\partial \dot{\mathbf{q}}} + \frac{\partial \Pi}{\partial \mathbf{q}} = \{\mathbf{Q}\} \quad (13.37)$$

where  $T$ ,  $\Pi$  – kinetic and potential energy of a mechanical system accordingly;  $\Phi$  – dissipation function;  $\{\mathbf{q}\}$ ,  $\{\dot{\mathbf{q}}\}$  – vectors of generalized coordinates and velocities accordingly;  $\{\mathbf{Q}\}$  – vector of generalized forces;  $t$  – time. The angles  $\varphi_1$ ,  $\varphi_2$  and  $\varphi_3$  are accepted as the generalized coordinates.

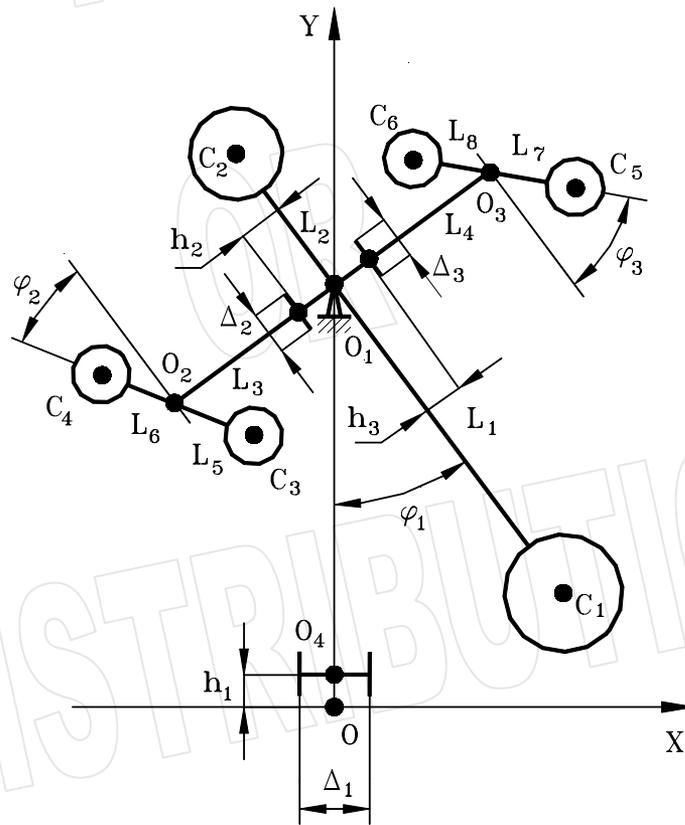


Fig. 13.11. The calculated scheme of explored mechanical system

The kinetic energy of a mechanical system is determined as follows:

$$T = \frac{1}{2} \sum_{i=1}^6 m_i v_{c_i}^2, \quad (13.38)$$

where  $m_i$  – masses of bodies and  $\mathbf{v}_{c_1} = L_1 \dot{\varphi}_1 (\cos \varphi_1 + \sin \varphi_1)$ ;  $\mathbf{v}_{c_2} = -L_2 \dot{\varphi}_1 (\cos \varphi_1 + \sin \varphi_1)$ ;

$$\mathbf{v}_{c_3} = \dot{\varphi}_1 [L_3 \sin \varphi_1 + L_5 \cos(\varphi_1 + \varphi_2)] + \dot{\varphi}_2 L_5 \cos(\varphi_1 + \varphi_2) + \dot{\varphi}_1 [-L_3 \cos \varphi_1 + L_5 \sin(\varphi_1 + \varphi_2)] + \dot{\varphi}_2 L_5 \sin(\varphi_1 + \varphi_2); \quad (13.39)$$

$$\mathbf{v}_{c_4} = \dot{\varphi}_1 [L_3 \sin \varphi_1 - L_6 \cos(\varphi_1 + \varphi_2)] - \dot{\varphi}_2 L_6 \cos(\varphi_1 + \varphi_2) - \dot{\varphi}_1 [L_3 \cos \varphi_1 + L_6 \sin(\varphi_1 + \varphi_2)] - \dot{\varphi}_2 L_6 \sin(\varphi_1 + \varphi_2);$$

$$\begin{aligned} v_{c_5} = & \dot{\varphi}_1[-L_4 \sin \varphi_1 + L_7 \cos(\varphi_1 + \varphi_3)] + \dot{\varphi}_3 L_7 \cos(\varphi_1 + \varphi_3) + \\ & + \dot{\varphi}_1[L_4 \cos \varphi_1 + L_7 \sin(\varphi_1 + \varphi_3)] + \dot{\varphi}_3 L_7 \sin(\varphi_1 + \varphi_3) ; \end{aligned}$$

$$\begin{aligned} v_{c_6} = & \dot{\varphi}_1[-L_3 \sin \varphi_1 - L_8 \cos(\varphi_1 + \varphi_3)] - \dot{\varphi}_3 L_8 \cos(\varphi_1 + \varphi_3) + \\ & + \dot{\varphi}_1[L_4 \cos \varphi_1 - L_8 \sin(\varphi_1 + \varphi_3)] - \dot{\varphi}_3 L_8 \sin(\varphi_1 + \varphi_3) . \end{aligned}$$

The *potential energy* of a mechanical system is defined by the next relation:

$$\Pi = 0 \quad (13.40)$$

The *dissipation function* of a mechanical system is defined as follows:

$$\Phi = \frac{1}{2} \sum_{i=1}^3 H_i \dot{\varphi}_i^2, \text{ where } H_i - \text{a damping coefficient} \quad (13.41)$$

The *work of force of own weight* is determined as follows:

$$W = \sum_{i=1}^6 \{F_{c_i}\}^T \delta\{r_{c_i}\} = \sum_{i=1}^6 \{F_{c_i}\}^T \left[ \frac{\partial r_{c_i}}{\partial \varphi_j} \right] \{\delta\varphi_j\}, \quad j = 1, 2, 3 \quad (13.42)$$

where  $\{F_{c_i}\}$  - vector of own weight of  $i$ -th mass,  $\{F_{c_i}\}^T = [0 \quad -m_i g \quad 0]$ ;  $\{r_{c_i}\}$  - vector of centre of masses

$$\text{of } i\text{-th body and } \{r_{c_1}\} = \begin{Bmatrix} L_1 \sin \varphi_1 \\ L_9 - L_1 \cos \varphi_1 \\ 0 \end{Bmatrix}; \{r_{c_2}\} = \begin{Bmatrix} -L_2 \sin \varphi_1 \\ L_9 + L_2 \cos \varphi_1 \\ 0 \end{Bmatrix};$$

$$\{r_{c_3}\} = \begin{Bmatrix} -L_3 \cos \varphi_1 + L_5 \sin(\varphi_1 + \varphi_2) \\ L_9 - L_3 \sin \varphi_1 - L_5 \cos(\varphi_1 + \varphi_2) \\ 0 \end{Bmatrix}; \{r_{c_4}\} = \begin{Bmatrix} -L_3 \cos \varphi_1 - L_6 \sin(\varphi_1 + \varphi_2) \\ L_9 - L_3 \sin \varphi_1 + L_6 \cos(\varphi_1 + \varphi_2) \\ 0 \end{Bmatrix}; \quad (13.43)$$

$$\{r_{c_5}\} = \begin{Bmatrix} L_4 \cos \varphi_1 + L_7 \sin(\varphi_1 + \varphi_3) \\ L_9 + L_4 \sin \varphi_1 - L_7 \cos(\varphi_1 + \varphi_3) \\ 0 \end{Bmatrix}; \{r_{c_6}\} = \begin{Bmatrix} L_4 \cos \varphi_1 + L_8 \sin(\varphi_1 + \varphi_3) \\ L_9 + L_4 \sin \varphi_1 - L_8 \cos(\varphi_1 + \varphi_3) \\ 0 \end{Bmatrix}$$

The work of magnetic forces acting onto the first body, is defined as follows:

$$W_{M, c_1} = F_{M, c_1} \delta l_{c_1, o_4} \quad (13.44)$$

where:  $F_{M, c_1}$  - force of a constant magnet acting onto the first body;

$$F_{M, c_1} = c e^{-a l_{c_1, o_4}}; \quad (13.45)$$

$c, a$  - constants;  $l_{c_1, o_4}$  - distance from a point  $C_1$  up to a point  $O_4$  (see fig. 13.11),

$$l_{c_1, o_4} = \left( \{R_{c_1, o_4}\}^T \{R_{c_1, o_4}\} \right)^{\frac{1}{2}}; \{R_{c_1, o_4}\} = \{r_{c_1}\} - \{r_{o_4}\}; \{r_{o_4}\}^T = [0 \quad h_1 \quad 0]. \quad (13.46)$$

The variation of distance  $l_{o_4, c_1}$  is defined by the following relation:

$$\delta I_{c_1, o_4} = \frac{\partial I_{c_1, o_4}}{\partial \{R_{c_1, o_4}\}} \delta \{R_{c_1, o_4}\} = \frac{1}{I_{c_1, o_4}} \{R_{c_1, o_4}\}^T \delta \{R_{c_1, o_4}\} = \frac{1}{I_{c_1, o_4}} \{R_{c_1, o_4}\}^T \left\{ \frac{\partial R_{c_1, o_4}}{\partial \varphi_1} \right\} \delta \varphi_1 \quad (13.47)$$

Then the generalized magnetic force acting onto the first body, is determined as follows:

$$W_{M, c_1} = F_{M, c_1} \frac{1}{I_{c_1, o_4}} \{R_{c_1, o_4}\}^T \left\{ \frac{\partial R_{c_1, o_4}}{\partial \varphi_1} \right\} \delta \varphi_1 = Q_{M, c_1} \delta \varphi_1, \quad (13.48)$$

$$\text{where } Q_{M, c_1} = \frac{F_{M, c_1}}{I_{c_1, o_4}} \{R_{c_1, o_4}\}^T \left\{ \frac{\partial R_{c_1, o_4}}{\partial \varphi_1} \right\} \quad (13.49)$$

The *generalized magnetic forces* acting onto remaining bodies, are similarly determined, namely:

$$Q_{M, c_3} = \frac{F_{M, c_3}}{I_{c_3, o_5}} \{R_{c_3, o_5}\}^T \left\{ \frac{\partial R_{c_3, o_5}}{\partial \varphi_2} \right\}; \quad Q_{M, c_4} = \frac{F_{M, c_4}}{I_{c_4, o_5}} \{R_{c_4, o_5}\}^T \left\{ \frac{\partial R_{c_4, o_5}}{\partial \varphi_2} \right\}; \quad (13.50)$$

$$Q_{M, c_5} = \frac{F_{M, c_5}}{I_{c_5, o_6}} \{R_{c_5, o_6}\}^T \left\{ \frac{\partial R_{c_5, o_6}}{\partial \varphi_3} \right\}; \quad Q_{M, c_6} = \frac{F_{M, c_6}}{I_{c_6, o_6}} \{R_{c_6, o_6}\}^T \left\{ \frac{\partial R_{c_6, o_6}}{\partial \varphi_3} \right\}$$

The *generalized magnetic forces* act onto bodies under next *conditions*:

$$-\frac{\Delta_1}{2} \leq L_1 \sin \varphi_1 \leq \frac{\Delta_1}{2}; \quad -\frac{\Delta_2}{2} \leq -L_5 \cos \varphi_2 \leq \frac{\Delta_2}{2}; \quad -\frac{\Delta_2}{2} \leq L_6 \cos \varphi_2 \leq \frac{\Delta_2}{2}; \\ -\frac{\Delta_3}{2} \leq -L_7 \cos \varphi_3 \leq \frac{\Delta_3}{2}; \quad -\frac{\Delta_3}{2} \leq -L_8 \cos \varphi_3 \leq \frac{\Delta_3}{2} \quad (13.51)$$

The *force of resistance* to moving of *i*-th body in air is determined as follows:

$$F_{\text{resist}, i} = \frac{1}{2} c \rho S v^2 \quad (13.52)$$

where *c* – form factor of a body (for sphere  $c \approx 0.5$ );  $\rho$  – density of air; *S* – frontal area of a body; *v* – absolute velocity of motion of a body. Then, the moment of resistance to moving of *i*-th body is determined by the next relation:

$$M_{\text{resist}, i} = [R_i] \{F_{\text{resist}, i}\} = \begin{bmatrix} -R_{y, i} & R_{x, i} \end{bmatrix} \begin{Bmatrix} F_{\text{resist}, x, i} \\ F_{\text{resist}, y, i} \end{Bmatrix}, \quad (13.53, 13.54)$$

where  $\{F_{\text{resist}, i}\}$  – vector of resistance forces;  $\{F_{\text{resist}, i}\} = -\frac{F_{\text{resist}, i}}{\sqrt{\dot{x}_{c_i}^2 + \dot{y}_{c_i}^2}} \begin{Bmatrix} \dot{x}_{c_i} \\ \dot{y}_{c_i} \end{Bmatrix}$ ;  $\dot{x}_{c_i}$ ,  $\dot{y}_{c_i}$  – projections of velocity of motion of a body in the frame *X* - *Y*;  $R_{x, i}$ ,  $R_{y, i}$  – components of radius-vector from a rotation axis up to centre of masses of a body;

$$R_{x, 1} = L_1 \sin \varphi_1; \quad R_{y, 1} = -L_1 \cos \varphi_1; \quad R_{x, 2} = -L_1 \sin \varphi_1; \quad R_{y, 2} = L_1 \cos \varphi_1; \quad R_{x, 3} = L_5 \sin(\varphi_1 + \varphi_2); \\ R_{y, 3} = -L_5 \cos(\varphi_1 + \varphi_2); \quad R_{x, 4} = -L_6 \sin(\varphi_1 + \varphi_2); \quad (13.55)$$

$$R_{y, 4} = L_6 \cos(\varphi_1 + \varphi_2); \quad R_{x, 5} = L_7 \sin(\varphi_1 + \varphi_3); \quad R_{y, 5} = -L_7 \cos(\varphi_1 + \varphi_3); \\ R_{x, 6} = -L_8 \sin(\varphi_1 + \varphi_3); \quad R_{y, 6} = L_8 \cos(\varphi_1 + \varphi_3)$$

### 13.4.2. Input data for the solution of the problem

For the solution of a problem over investigation of *motion of the mechanical system* represented on the *fig. 13.12*, the *Pendulum* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable **F**, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the following parameters are coded: print code of intermediate results (*kprint*), number of integration steps (*ntime*), number indicating through how much of integration steps the results of evaluations are recorded (*nstep*), and integration step (*dtime*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. In the *second* line, a line number and nine geometrical parameters  $L_i$  ( $i=1, \dots$ ) of array **AL(9)** are coded.

In subsequent *six* lines, the number of a body and elements of arrays **BM(6)**, **R(6)** and **AH(6)** are coded. Into these arrays a mass, radius and a damping coefficient of each ball are recorded. In subsequent three lines are coded: the number of a constant magnet and also elements of arrays **DD(3)** and **DH(3)**, into which a *thickness* of a magnet (*parameter*  $\Delta$ ) and *distance* up to a magnet from a point  $O_i$  (*parameter*  $h_i$ ) are recorded accordingly. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

#### Schematic structure of the datafile:

Text line \*  
*kprint, ntime, nstep, dtime*  
 Text line \*  
 Array **AL(7)**  
 Text line \*  
 Arrays **BM(6), R(6), AH(6)**  
 Text line \*  
 Arrays **DD(3), DH(3)**  
 Text line \*  
 Array **Xvarb(6)**

### 13.4.3. Brief description of the Pendium program solving the problem

The *Pendium* program was programmed on the *Maple*-language; it consists of the basic program and 23 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculation and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of integration steps. The program *Pendium* calculates the *generalized* coordinates and velocities of motion of the mechanical system represented on the *fig. 13.11*, and also provides animation of a resultant motion of the system. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1* and *file\_rez2* of the program must be ascribed *qualifiers* of target files. Into the file *file\_rez1* values of the *generalized* coordinates and velocities of motion are recorded. While into the file *file\_rez2* the expressions for equations of motion of a mechanical system with the concentrated parameters are recorded.

### 13.4.4. An example of use of the Maple-program Pendium

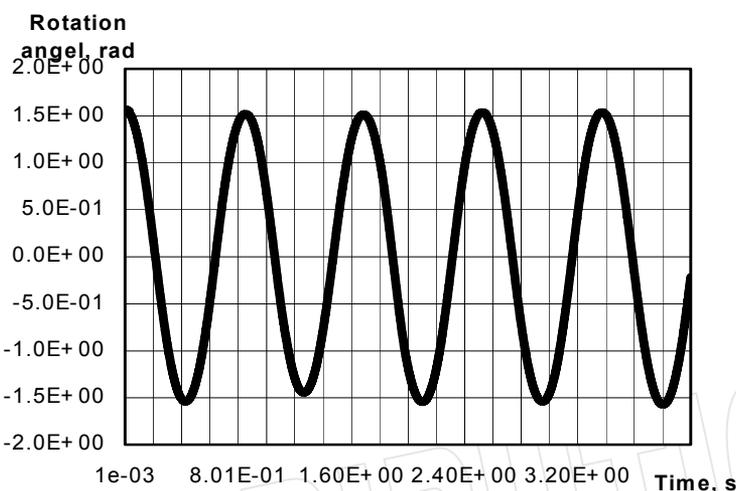
As an example of application of the *Pendium* program, the generalized coordinates and velocities of motion of the mechanical system indicated on the *fig. 13.11*, are determined.

Input data for the test example: a print code *kprint*=0, number of integration steps *ntime*=5000, *nstep* = 50, integration step *dtime*= $10^{-3}$  s. Geometrical parameters:

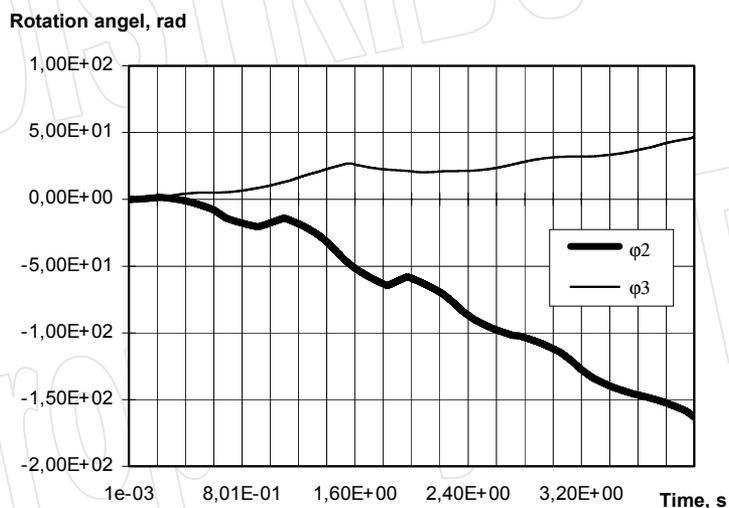
$AL(1) = 0,070 \text{ m}$  ;  $AL(2) = 0,085 \text{ m}$  ;  $AL(3) = AL(4) = 0,040 \text{ m}$  ;  
 $AL(5) = AL(6) = AL(7) = AL(8) = 0,020 \text{ m}$  ;  $AL(9) = 0,0975 \text{ m}$  ;  $BM(1) = 0,050 \text{ kg}$  ;  
 $BM(2) = 0,010 \text{ kg}$  ;  $BM(3) = BM(4) = BM(5) = BM(6) = 0,005 \text{ kg}$  ;  $R(1) = 0,0125 \text{ m}$  ;  
 $R(1) = 0,050 \text{ m}$  ;  $R(3) = R(4) = R(5) = R(6) = 0,010 \text{ m}$  ;  
 $AH(1) = AH(2) = AH(3) = AH(4) = AH(5) = AH(6) = 0$  ;  $DD(1) = DD(2) = DD(3) = 0,005 \text{ m}$  ;  
 $DH(1) = 0,005 \text{ m}$  ;  $DH(2) = DH(3) = 0,010 \text{ m}$  ;  $Xvarb(1) = 0,005 \text{ }^\circ$  ;  
 $Xvarb(2) = Xvarb(3) = Xvarb(4) = Xvarb(5) = Xvarb(6) = 0$  .

*Results of calculation over the test example:*

The dependences of rotation angles of masses and rotation velocities in a time are represented on the figs. 13.12 and 13.13, accordingly. On the fig. 13.14 four fragments (a, b, c, d) of motion of the explored mechanical system are represented.



(a)



(b)

Fig. 13.12. Dependences of rotation angles of masses of the explored mechanical system: (a) – angle  $\varphi_1$ , and (b) – angles  $\varphi_2$  and  $\varphi_3$

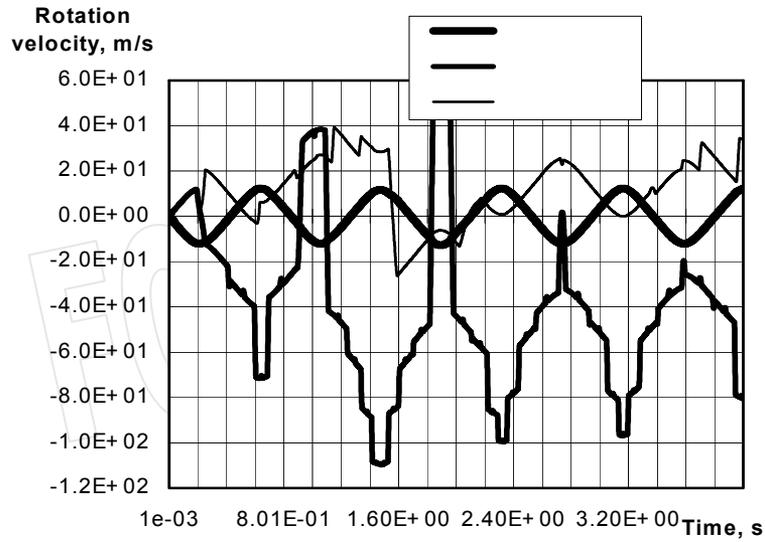


Fig. 13.13. Dependences of rotation velocities of masses of the explored mechanical system

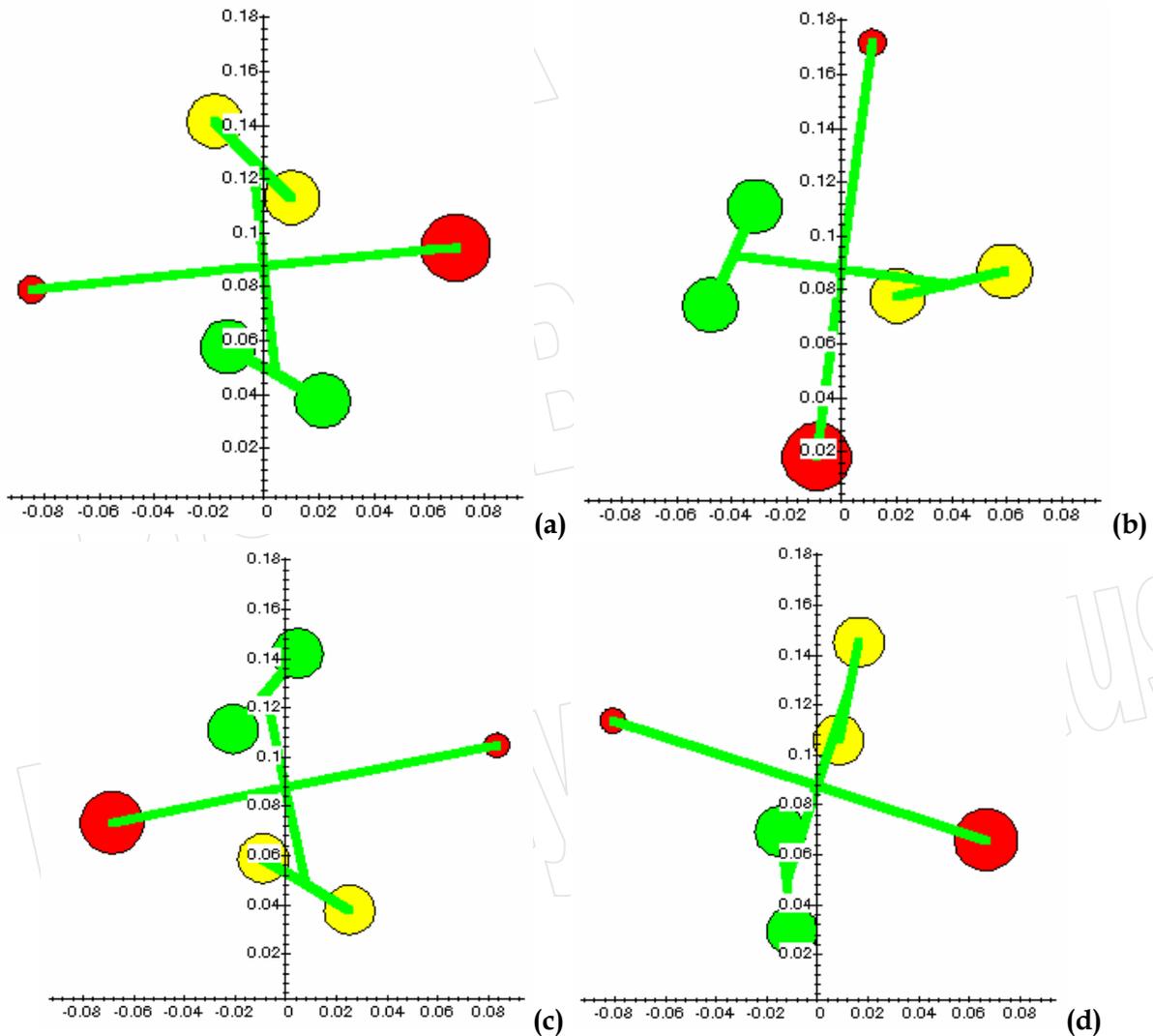


Fig. 13.14. Fragments of motion of the mechanical system

The obtained equations of motion of the explored system have the next form:

$$Eq_1 = .000357250 u_{tt1} + .400000 10^{-5} u_{tt2} + .400000 10^{-5} u_{tt3} + h_1 u_{t1} + .0259965 \sin(u_1) - 1. Qp_1 - 1. Qm_1 - 1. Qp_2 - 1. Qm_2$$

$$Eq_2 = .400000 10^{-5} u_{tt1} + .400000 10^{-5} u_{tt2} + h_2 u_{t2} - 1. Qp_3 - 1. Qm_3 - 1. Qp_4 - 1. Qm_4$$

$$Eq_3 = .400000 10^{-5} u_{tt1} + .400000 10^{-5} u_{tt3} - 1. Qp_5 - 1. Qm_5 - 1. Qp_6 - 1. Qm_6 ,$$

where  $\mathbf{u}$  - displacement;  $\mathbf{u}_t$  - velocity;  $\mathbf{u}_{tt}$  - acceleration;  $\mathbf{h}$  - coefficient of damping;  $\mathbf{Qp}$  - generalized force of resistance to motion of the  $i$ -th body in air;  $\mathbf{Qm}$  - generalized magnetic force acting onto a body. The *Pendium* program is intended for calculation of the *generalized* coordinates and velocities of motion of the mechanical system represented on the *fig. 13.11*, for deduction of equations of motion of the mechanical system with the concentrated parameters, and also for *animation* of motion of the explored mechanical system. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in files *Mb10\_4\_new.mws*, *Mb10\_4.dat* and *Mb10\_4\_1.rez* accordingly.

### 13.5. Calculation of transition torsion oscillations in a mechanical transmission

In technique the mechanical transmissions consisting from gears and shafts, which transmit a torsional moment from the drive to an operating mechanism, are widely used. The reason of torsional oscillations in shafts of transmission systems consists in the *nonuniformity* of the torsional moments from propellents and forces of resistance, which cause changes of angular velocity of rotation of the shaft, i.e., now acceleration, now deceleration of its rotation. The shaft has elasticity and the *concentrated masses* are placed on it, therefore in each cross-section of the shaft takes place own degree of nonuniformity.

The basic noisemaker and vibration in transmission systems (*for example, in transmissions*) is the knock of gears, which is characterized by vibrational blows in engagement of gears, which arise owing to a presence of a backlash between cogs. As a result of torsional oscillations, cyclic angular accelerations of details of a system of gears arise. When the torsional moment of inertia of a driven gear exceeds the moment of a loading, cogs of gears go away and blows arise. The continuous shocks called as vibrational blows, can reduce to excessive noise and major dynamical loadings onto cogs. Therefore, the calculation of torsional oscillations of mechanical transmissions consisting of the drive, shafts and gears with backlashes, represents major practical interest.

#### 13.5.1. The calculated expressions for determination of torsional oscillations of mechanical transmissions

A mechanical transmission consisting of the asynchronous electric motor, coupler, shafts and gears is considered. The influence of the drive onto the dynamics of all driving gear can be considerable, therefore in dynamic model of a driving gear it is necessary to take into account the influence of the drive. At investigation of asynchronous drives the two-phase mathematical models adequately reflecting processes happening in the drive are used. The common form of *mathematical* description of two-phase models is the system of the differential and algebraic equations [110], namely:

$$\left\{ \frac{dx}{dt} \right\} = [A]\{x\} + \{F(\psi, t)\}; \quad \frac{d\omega}{dt} = \frac{(M_{eng} - M_{resist}) p}{I_n}; \quad (13.56)$$

$$M_{\text{eng}} = 1,5pL_{\mu}a(\psi_{\alpha s}\psi_{\beta r} - \psi_{\alpha r}\psi_{\beta s}); \quad [A] = \begin{bmatrix} -ar_sL_r & 0 & ar_sL_{\mu} & 0 \\ 0 & -ar_sL_r & 0 & ar_sL_{\mu} \\ ar_rL_{\mu} & 0 & -ar_rL_s & 0 \\ 0 & ar_rL_{\mu} & 0 & -ar_rL_s \end{bmatrix}; \quad \{x\} = \begin{Bmatrix} \psi_{\alpha s} \\ \psi_{\beta s} \\ \psi_{\alpha r} \\ \psi_{\beta r} \end{Bmatrix};$$

$$\{F(\psi, t)\} = \begin{Bmatrix} \sqrt{2} U_n \cos \omega_s t \\ -\sqrt{2} U_n \sin \omega_s t \\ \omega \psi_{\beta r} \\ -\omega \psi_{\alpha r} \end{Bmatrix}, \text{ where } a = \frac{1}{L_s L_r - L_{\mu}^2}; \quad L_{\mu} = 1,5 L_0; \quad L_s = L_{s\sigma} + 1,5 L_0;$$

$L_r = L_{r\sigma} + 1,5 L_0$ ;  $L_{s\sigma}$ ,  $L_{r\sigma}$  - diffusion inductance of stator and rotor;  $L_0$  - inductance of a magnetizing contour;  $r_s$ ,  $r_r$  - active resistance of stator and rotor;  $U_n$  - rated voltage;  $\omega_s$  - angular frequency of a supply voltage;  $M_{\text{eng}}$  - moment of drive;  $p$  - number of pairs of poles;  $\omega$  - angular frequency of rotation of drive rotor;  $I_n$  - moment of inertia.

A mathematical model of torsional oscillations of a mechanical transmission with the concentrated parameters is considered. The model represents a system consisting of shafts, coupler and gears. In the model of the general view there are three nonlinear elements of stiffness: many-stage stiffness of a coupler, backlash between splines of coupler and a backlash between cogs of a pair gears. The common nonlinear element  $f_i(\delta_i)$  is represented on the following fig. 13.15.

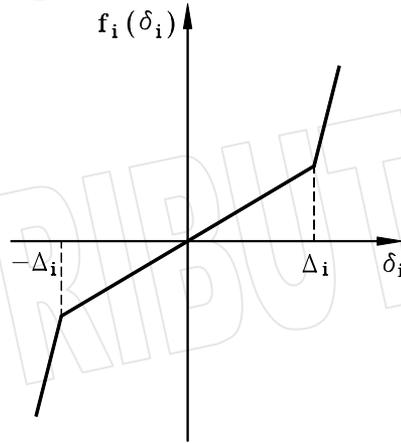


Fig. 13.15. Dependence of nonlinear force  $f_i(\delta_i)$  from displacement  $\delta_i$

The equations of motion of basic elements of a mechanical drive unit, are considered, namely: coupler (fig. 13.16.a):

$$\begin{aligned} I_i \ddot{\varphi}_i &= k_{i-1,i} f_{i-1}(\delta_{i-1}) R_{i,1} - k_{i,i+1} f_i(\delta_i) R_{i,2} - h_{i-1,i} (\dot{\varphi}_i - \dot{\varphi}_{i-1}) - h_{i,i+1} (\dot{\varphi}_i - \dot{\varphi}_{i+1}); \\ I_{i+1} \ddot{\varphi}_{i+1} &= k_{i,i+1} f_i(\delta_i) R_{i+1,2} - k_{i+1,i+2} f_{i+1}(\delta_{i+1}) R_{i+1,1} - h_{i,i+1} (\dot{\varphi}_{i+1} - \dot{\varphi}_i) - \\ &\quad - h_{i+1,i+2} (\dot{\varphi}_{i+1} - \dot{\varphi}_{i+2}); \end{aligned}$$

$$\delta_{i-1} = \varphi_{i-1} R_{i-1,1} - \varphi_i R_{i,1}; \quad \delta_i = \varphi_i R_{i,2} - \varphi_{i+1} R_{i+1,1}; \quad \delta_{i+1} = \varphi_{i+1} R_{i+1,1} - \varphi_{i+2} R_{i+2,1}, \quad (13.57)$$

where:  $I_i$ ,  $I_{i+1}$  - moment of inertia of a half-coupler;  $k_{i-1,i}$ ,  $k_{i+1,i+2}$  - stiffnesses of shafts;  $k_{i,i+1}$  - stiffness of a coupler;  $h_{i-1,i}$ ,  $h_{i,i+1}$ ,  $h_{i+1,i+2}$  - coefficients of damping;  $\varphi_i$ ,  $\dot{\varphi}_i$ ,  $\ddot{\varphi}_i$  - angular displacements, velocities and accelerations accordingly;  $\delta_i$  - relative displacements;  $R_{i,1}$ ,  $R_{i+1,1}$  - radiuses of dividing

splices of a coupler;  $R_{i,2}$ ,  $R_{i+1,2}$  – the radiuses of circles, on which are located springs of a coupler.

The nonlinear function  $f_i(\delta_i)$  of the coupler is defined by the next relation:

$$f_i(\delta_i) = \begin{cases} \delta_i - \Delta_i(1 - \alpha_i), & \delta_i > \Delta_i; \\ \alpha_i \delta_i, & -\Delta_i \leq \delta_i \leq \Delta_i; \\ \delta_i + \Delta_i(1 - \alpha_i), & \delta_i < -\Delta_i, \end{cases} \quad (13.58)$$

where:  $\Delta_i$  – salient point for steps of nonlinearity;  $\alpha_i$  – measure of nonlinearity.

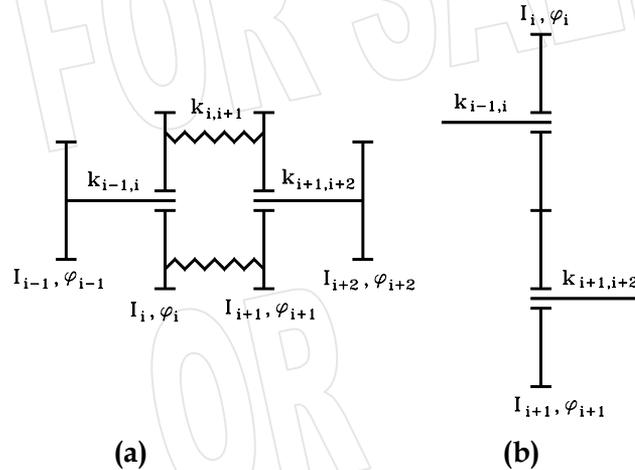


Fig. 13.16. Basic elements of mechanical transmission: (a) a coupler and (b) a pair of gears pair of gears (fig. 13.16.b):

$$I_i \ddot{\varphi}_i = k_{i-1,i} f_{i-1}(\delta_{i-1}) R_{i,1} - k_{i,i+1} f_i(\delta_i) R_{i,2} - h_{i-1,i} (\dot{\varphi}_i - \dot{\varphi}_{i-1}); \quad (13.59)$$

$$I_{i+1} \ddot{\varphi}_{i+1} = k_{i,i+1} f_i(\delta_i) R_{i+1,2} - k_{i+1,i+2} f_{i+1}(\delta_{i+1}) R_{i+1,1} - h_{i+1,i+2} (\dot{\varphi}_{i+1} - \dot{\varphi}_{i+2}),$$

where:  $I_i$ ,  $I_{i+1}$  – moment of inertia of gears;  $k_{i,i+1}$  – stiffness of a hitch;  $h_{i-1,i}$ ,  $h_{i,i+1}$ ,  $h_{i+1,i+2}$  – coefficients of damping of shafts;  $R_{i,1}$ ,  $R_{i+1,1}$  – radiuses of dividing circles of splines;  $R_{i,2}$ ,  $R_{i+1,2}$  – radiuses of dividing circles of a leading and driven gears. The nonlinear function  $f_i(\delta_i)$  for a backlash between cogs is defined as follows:

$$f_i(\delta_i) = \begin{cases} \delta_i - \Delta_i(1 - \alpha_i), & \delta_i > \Delta_i; \\ 0, & -\Delta_i \leq \delta_i \leq \Delta_i; \\ \delta_i + \Delta_i(1 - \alpha_i), & \delta_i < -\Delta_i. \end{cases} \quad (13.60)$$

Generally, the external torsional moment is defined by the following relation:

$$M_{\text{resist}} = M_{\text{resist0}} + M_{\text{resist1}} \sin(\omega_{\text{resist}} t), \quad (13.61)$$

where:  $M_{\text{resist0}}$ ,  $M_{\text{resist1}}$  – constant and variable part of a torsional moment. The parameters  $M_{\text{resist0}}$ ,  $M_{\text{resist1}}$  and  $\omega_{\text{resist}}$  can be by constant values or independent random quantities, distributed under the normal law. The value of a random quantity  $y$  submitting to the normal distribution, is defined as follows:

$$y = \mu + \sigma \left( \sum_{i=1}^{12} R_i - 6 \right), \quad (13.62)$$

where:  $\mu$  – average value;  $\sigma^2$  – variance;  $R_i$  – random number from the interval  $[0, 1]$ . Then the common equations system of motion of the mechanical transmission with an electromotor accepts the following form:

$$\{\dot{\mathbf{x}}\} = [\mathbf{A}]\{\mathbf{x}\} + \{\mathbf{F}(t, \mathbf{x})\}; \quad [\mathbf{M}]\{\ddot{\phi}\} = \{\mathbf{G}(t, \phi, \dot{\phi}, \mathbf{x})\} \quad (13.63)$$

We solve a problem of eigenvalues of system of the linear equations without taking the equations of the electromotor into consideration.

### 13.5.2. Input data for the solution of the problem

For solution of a problem concerning of *definition of torsion oscillations of mechanical transmissions* the *Driver* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable **F**, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely.

In the *first* line the next parameters are recorded: number of bodies of a mechanical transmission (*parameter nbody*); the number of elements linking bodies of mechanical transmission (*nelem*); the number of the moments of resistance (*nmpas*); the number of degrees of freedom whose values of unknowns are recorded into the resultant file (*nwr*) and code of a loading (*kodpas*). If *kodpas*=1, then takes place a casual loading; otherwise – determinate loading.

In the *second* line the next parameters are recorded: a print code of intermediate results (*kprint*); number indicating through how much of integration steps the results of evaluations are recorded (*nstep*); number of iterations steps (*ntime*) and iteration step (*dtime*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out.

In each subsequent *nelem* lines, the elements of the array **Mtop**(*nelem*, 5) are coded. Into each line of the datafile the line and five elements of this array are recorded, namely: the number of the first and the second body, which are connected among themselves by an element of transmission, and also three numbers (0 or 1), which encode a type of linking of the bodies. The first number defines a splice linking, the second – a cog linking, the third – a coupler. The number 1 or 0 indicates presence of linking or its absence accordingly.

Behind of the array **Mtop** into the datafile the elements of the **Lpar**(*nbody*, 2) are recorded line by line. Into each line of the datafile the line number and elements of the array **Lpar** are recorded. Into each line of the array **Lpar** the code defining number of a body to which an electromotor is connected, and number of the external moment of resistance are recorded. If the electromotor is connected to the corresponding body, the code equals 1, otherwise – 0. If onto the corresponding body the external moment of resistance does not act, then into the *second* column of the array **Lpar** the value 0 is recorded; otherwise, the serial number of the external moment of resistance is recorded.

Behind of the array **Lpar** into the datafile the elements of the **Par**(*nbody*, 10) array are recorded line by line. Into each line of the datafile the line number and elements of the **Par** array are recorded. Into each line of this array are recorded the following parameters: moment of inertia of mass  $I_i$ ; radius of dividing circle of splices  $R_{i,1}$ ; a salient point of a linking  $\Delta_{i,1}$ ; a measure of nonlinearity  $\alpha_{i,1}$  of splice linking; radius of dividing circle of gears  $R_{i,2}$ ; salient point of linkings  $\Delta_{i,2}$ ; a measure of nonlinearity  $\alpha_{i,2}$  of cog linking; radius of a circle, on which the springs of a coupler  $R_{i,3}$  are located; a salient point of linking  $\Delta_{i,3}$  and also measure of nonlinearity  $\alpha_{i,3}$  of coupler linking.

Behind of the array **Par** into the datafile the elements of the **Par1**(*nelem*, 2) array are recorded line by line. Into each line of the datafile the line number and elements of the **Par1** array are recorded. Into each line of this array the coefficients of stiffness and damping of an element of linking are recorded.

Behind of the array **Par1** into the datafile the elements of the **Par\_m**(*nmpas*, 6) array are recorded line by line. Into each line of the datafile the line number and elements of the **Par\_m** array are recorded. Into each line of this array are recorded the following parameters: the line number; average value and mean square deviation of a constant component of the moment of resistance; average value and mean square deviation of amplitude of the moment of resistance; average value and mean square deviation of frequency of the moment of resistance. The line number of the array **Par\_m** must correspond to the number of an linkage element.

Behind of the array **Par\_m** into the datafile the elements of the **Par\_v**(7) array are recorded line by line. Into each line of the datafile the line number and elements of the **Par\_v** array are recorded. Into the array **Par\_v** the parameters of an asynchronous electromotor are recorded, namely: rated voltage **U<sub>n</sub>**; number of pairs of poles **p**; an inductance dispersion of a stator winding **L<sub>sσ</sub>**; the reduced inductance dispersion of a rotor winding **L<sub>rσ</sub>**; inductance of a magnetizing contour **L<sub>0</sub>**; active resistance of stator **r<sub>s</sub>** and rotor **r<sub>r</sub>**.

Behind of the array **Par\_v** into the datafile the elements of the **Mwr**(*nwr*) array are recorded line by line. In each line of the datafile the line number and also the number of a degree of freedom (*number of a body*), the values of angular displacements, velocities and accelerations of which are recorded into the resultant file, are coded.

Behind of the array **Mwr** into the datafile the elements of the array **Xvarb**(*neq*) {*neq* – number of unknowns; *neq=4+2\*nbody*} are recorded line by line. In each line of the datafile the line number of the array and also an initial value of an unknown are coded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nbody, nelem, nmpas, nwr, kodpas, kprint, nstep, ntime, dtime*  
Text line \*  
Array **Mtop**(*nelem*, 5)  
Text line \*  
Array **Lpar**(*nbody*, 2)  
Text line \*  
Array **Par**(*nbody*, 10)  
Text line \*  
Array **Par1**(*nelem*, 2)  
Text line \*  
Array **Par\_m**(*nmpas*, 6)  
Text line \*  
Array **Par\_v**(7)  
Text line \*  
Array **Mwr**(*nwr*)  
Text line \*  
Array **Xvarb**(*neq*)

### 13.5.3. Brief description of the Driver program solving the problem

The *Driver* program was programmed on the *Maple*-language; it consists of the basic program and 17 procedures. All procedures can be divided into two groups: procedures for data entry and for calculations. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of bodies of the concrete mechanical transmission. The program calculates values of angular displacements, velocities and accelerations of bodies of a mechanical transmission in a time dependence, and also eigenvalues and own frequencies. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3*, *file\_rez4* and *file\_rez5* of the program must be ascribed the qualifiers of target files. Into these files the following data are recorded: into *file\_rez1* – values of *angular displacements* of bodies of a mechanical transmission, into *file\_rez2* – values of *angular velocities*, into *file\_rez3* – values of *angular accelerations*, into *file\_rez4* – values of of a *rotational moment* of the electromotor, and into *file\_rez5* – *eigenvalues* and *own frequencies*.

### 13.5.4. An example of use of the Maple-program Driver

As an example of application of the *Driver* program, the mechanical transmission represented on the fig. 13.17 and consisting from asynchronous electromotor 4A100/4SY3, of a coupler and two cog transmissions is considered.

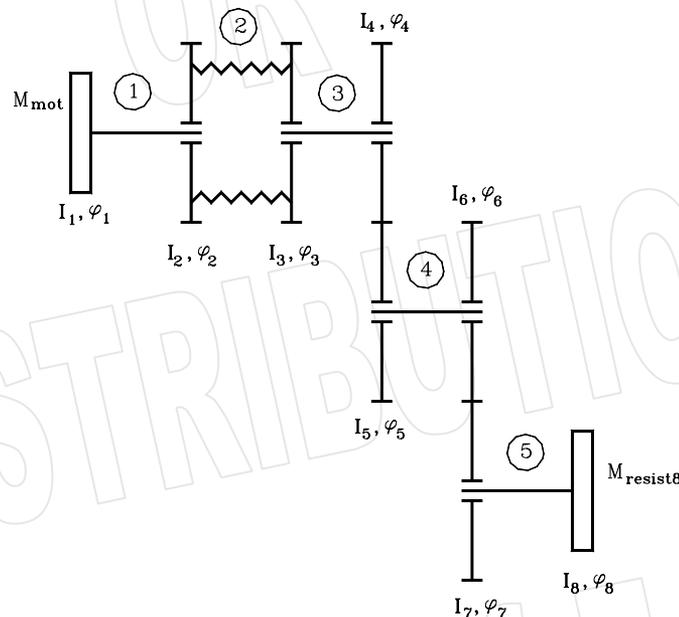


Fig. 13.17. The scheme of explored mechanical transmission

Input data for the test example: number of the bodies *nbody*=8; the number of elements linking the bodies *nelem*=5; number of nodes *npoin*=44; number of the moments of resistance *nmpas*=1; number of degrees of freedom, whose values of unknowns are recorded into the resultant file *nwr*=3; a code of a loading *kodpas*=0; *kprint*=1; *nstep*=2; *ntime*=5000 and *dtime*= $2 \cdot 10^{-4}$  s. The moment of resistance acting to the eighth body of the system:

$M_{resist0} = 50 \text{ Nm}$  ,  $M_{resist1} = 10 \text{ Nm}$  ,  $\omega_{resist0} = 62,8 \text{ s}^{-1}$ . Coefficients of elements stiffness:  
 $k_1 = 188,4 \text{ MN/m}$  ;  $k_2 = 1,02 \text{ MN/m}$  ;  $k_3 = 110,0 \text{ MN/m}$  ;  $k_4 = 83,9 \text{ MN/m}$  ;  
 $k_5 = 323,3 \text{ MN/m}$  ;  $k_6 = 146,4 \text{ MN/m}$  ;  $k_7 = 8,28 \text{ MN/m}$  . Coefficients of elements damping:  
 $h_i = 10^{-5} \text{ N s/m}$  ,  $i=1..7$ . Initial values of unknowns:  $x_i = \dot{x}_i = 0$  ,  $i=1..7$ . Backlashes of the size:

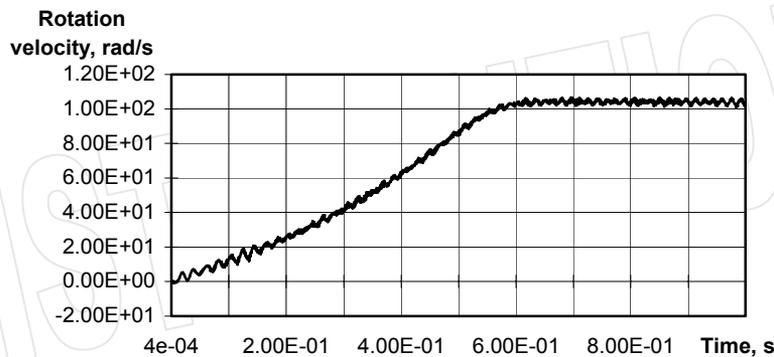
$\Delta_{6-7} = 10^{-4} \text{ m}$  ;  $\Delta_8 = 2 \cdot 10^{-4} \text{ m}$  . Measures of nonlinearity:  $\alpha_{6-7} = 0$  ;  $\alpha_8 = 0$  .

**Results of calculation over the test example:**

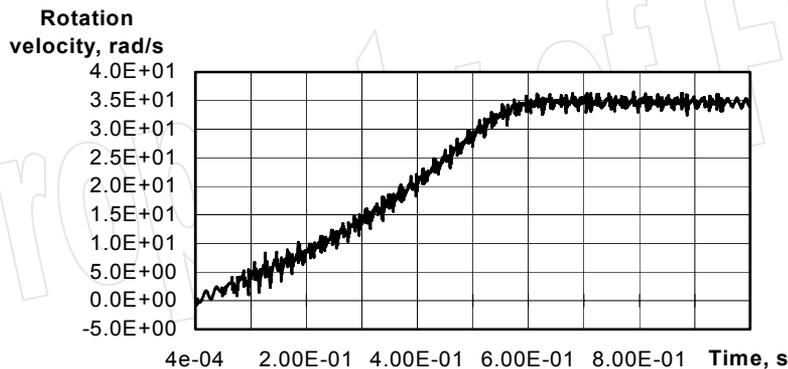
*Eigenvalues (real and imaginary part) and own frequencies:*

Re[1]= -2.817913e-06	Im[1]=8.965371e+03	frequency[1]=1.506968e+01 Hz
Re[2]= -2.817913e-06	Im[2]=-8.965371e+03	frequency[2]=1.506968e+01 Hz
Re[3]= -1.660601e-06	Im[3]=6.102317e+03	frequency[3]=1.243275e+01 Hz
Re[4]= -1.660601e-06	Im[4]=-6.102317e+03	frequency[4]=1.243275e+01 Hz
Re[5]= -1.149593e-06	Im[5]=2.763744e+03	frequency[5]=8.366986e+00 Hz
Re[6]= -1.149593e-06	Im[6]=-2.763744e+03	frequency[6]=8.366986e+00 Hz
Re[7]= -2.51011e-06	Im[7]=2.613511e+03	frequency[7]=8.136400e+00 Hz
Re[8]= -2.51011e-06	Im[8]=-2.613511e+03	frequency[8]=8.136400e+00 Hz
Re[9]= -1.207071e-06	Im[9]=2.080419e+03	frequency[9]=7.259313e+00 Hz
Re[10]= -1.207071e-06	Im[10]=-2.080419e+03	frequency[10]=7.259313e+00 Hz
Re[11]= 2.799271e-18	Im[11]=0e-01	frequency[11]=0e-01 Hz
Re[12]= -3.939402e-07	Im[12]=3.699337e+02	frequency[12]=3.061132e+00 Hz
Re[13]= -3.939402e-07	Im[13]=-3.699337e+02	frequency[13]=3.061132e+00 Hz
Re[14]= -3.605612e-07	Im[14]=4.320143e+02	frequency[14]=3.308028e+00 Hz
Re[15]= -3.605612e-07	Im[15]=-4.320143e+02	frequency[15]=3.308028e+00 Hz
Re[16]= 0e-01	Im[16]=0e-01	frequency[16]=0e-01 Hz

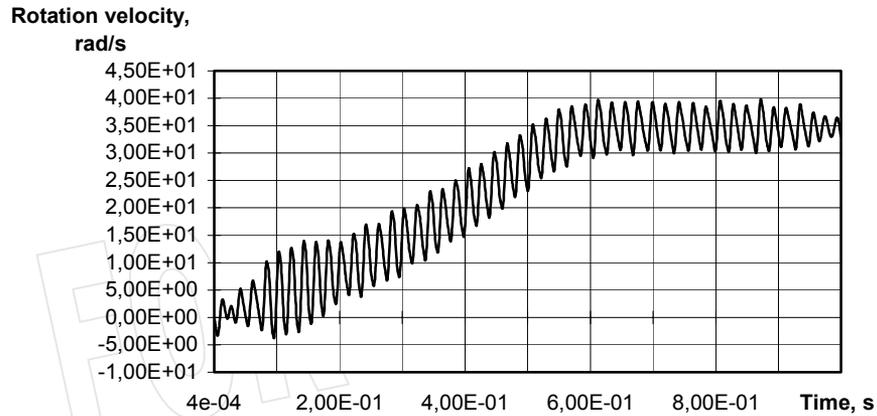
The fig. 13.18 represents the dependences of angular velocities of bodies 6, 7 and 8.



(a)



(b)



(c)

Fig. 13.18. Dependences of angular velocities of the body 6 (a), 7 (b) and 8 (c) accordingly

The *Driver* program is intended for the solution of a problem over definition of torsion oscillations of mechanical transmissions; the angular displacements, velocities and accelerations of bodies in a time dependence, and also *eigenvalues* and *own frequencies* are calculated. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files Mb10\_5\_new.mws, Mb10\_5.dat and Mb10\_5\_1.rez accordingly.

# Chapter 14.

## The applied problems of mechanics - 2

The problems of motion of vehicles over rough way present major applied interest. Most of the problems associated with the safety, economy, and overall quality of road transportation are affected by the characteristics of both roads and vehicles and by the manner in which these two dynamic systems interact. Unlike other publications that deal with either vehicles or roads, the present chapter places equal emphasis on the vehicle and the road. Motion of vehicle on rough road is considered as an example of the application of the FEM with using of mathematical package *Maple* for investigation of vehicle dynamics. At a motion of vehicles over a rough road onto elements of a suspension the major loadings affect, which in turn decrease longevity of elements of the suspension and the vehicle as a whole. At designing of vehicles the major attention is given to their comfortability. Generally, the kinematic excitation is a random loading acting onto a vehicle. Therefore, the problem solving at a random kinematic excitation also represents practical interest. In modern vehicles the pneumatic systems are widely applied. The modelling of one of such type of systems is considered in the present chapter.

### 14.1. Motion of a vehicle on a rough road

The oscillations of a vehicle at its motion on a rough road have an influence upon a state of the driver and passengers. The oscillations of the vehicle are reflected in safety of a transported load and vehicle itself. Therefore, by one of the basic properties demanded from a modern vehicle, are increase of smoothness of a motion and improving of motion comfortability. The oscillations of the vehicle are divided onto low-frequency oscillations (*up to 15-18 Hz*), onto high-frequency oscillations and vibrations. The vibrating sensitivity of a man organism lies in diapason 15-1500 Hz. With a high frequency the *sprung masses* vibrate, while with low - *unsprung masses* (*for example, body*). The activity of oscillations onto a man organism depends on such factors as: frequency, amplitude, duration of activity and direction. The influence of alternating accelerations onto man organism, in the greater degree, depends on an oscillation frequency. With magnification of a frequency even the small accelerations of oscillations can cause unpleasant sensations. The motion of a vehicle over the given profile of unevennesses of a road and with certain velocity of a motion is considered.

#### 14.1.1. The calculated expressions for definition of motion of a vehicle

The motion of a vehicle, consisting from a cabin, driver, frame, body, suspension and busbars, in the vertical plane is considered. The calculated scheme of a vehicle is represented on the *fig. 14.1*.

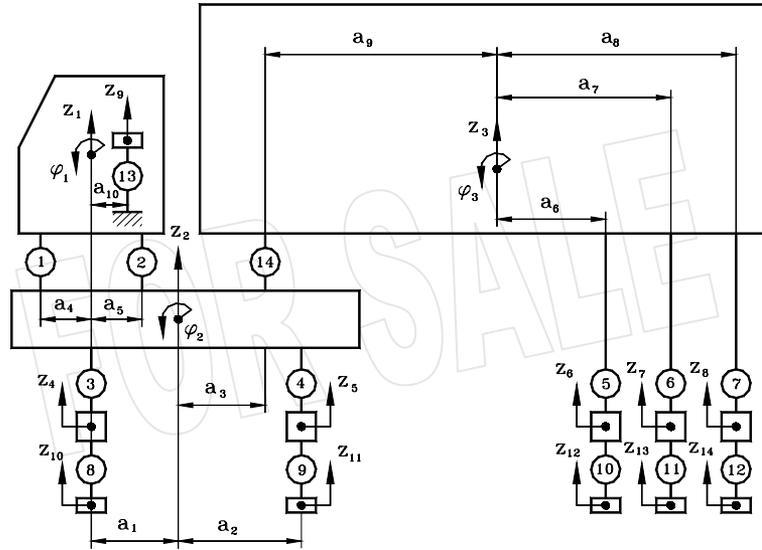


Fig. 14.1. The calculated scheme of a vehicle (by the circles are marked the elements, consisting of a spring and a damper)

The equations of motion of a vehicle are deduced on the basis of the Lagrange equation of the second type, namely:

$$\frac{d}{dt} \left( \frac{\partial T}{\partial \dot{q}_i} \right) - \frac{\partial T}{\partial q_i} + \frac{\partial \Phi}{\partial \dot{q}_i} + \frac{\partial \Pi}{\partial q_i} = Q_i \quad (14.1)$$

where  $T$ ,  $\Pi$  – kinetic and potential energy of a vehicle;  $\Phi$  – dissipation function;  $q_i$ ,  $\dot{q}_i$  – vectors of the generalized coordinates and velocities;  $Q_i$  – vector of the generalized force;  $t$  – time. Then the kinetic energy of the vehicle is defined by the following relation:

$$T = \frac{1}{2} \sum_{i=1}^3 (m_i \dot{z}_i^2 + I_i \dot{\varphi}_i^2) + \frac{1}{2} \sum_{i=4}^{14} m_i \dot{z}_i^2 \quad (14.2)$$

where  $m_i$ ,  $I_i$  – mass and moment of inertia of mass of the  $i$ -th body;  $\dot{z}_i$ ,  $\dot{\varphi}_i$  – the generalized velocities of the  $i$ -th body. The potential energy of the vehicle is defined by the following relation:

$$\begin{aligned} \Pi &= \sum_{i=1}^{14} \Pi_i, \text{ where: } \Pi_1 = \frac{1}{2} k_1 [(z_2 - a_4 \varphi_1) - (z_2 - (a_1 + a_4) \varphi_2)]^2; \\ \Pi_2 &= \frac{1}{2} k_2 [(z_1 + a_5 \varphi_1) - (z_2 - (a_1 - a_5) \varphi_2)]^2; \quad \Pi_3 = \frac{1}{2} k_3 [z_4 - (z_2 - a_1 \varphi_2)]^2; \\ \Pi_4 &= \frac{1}{2} k_4 [z_5 - (z_2 - a_1 \varphi_2)]^2; \quad \Pi_5 = \frac{1}{2} k_5 [(z_3 + a_6 \varphi_3) - z_6]^2; \\ \Pi_6 &= \frac{1}{2} k_6 [(z_3 + a_7 \varphi_3) - z_7]^2; \quad \Pi_7 = \frac{1}{2} k_7 [(z_3 + a_8 \varphi_3) - z_8]^2; \quad \Pi_8 = \frac{1}{2} k_8 (z_4 - z_8)^2; \\ \Pi_9 &= \frac{1}{2} k_9 (z_5 - z_{11})^2; \quad \Pi_{10} = \frac{1}{2} k_{10} (z_6 - z_{12})^2; \quad \Pi_{11} = \frac{1}{2} k_{11} (z_7 - z_{13})^2; \\ \Pi_{12} &= \frac{1}{2} k_{12} (z_8 - z_{14})^2; \quad \Pi_{13} = \frac{1}{2} k_{13} [z_9 - (z_1 + a_{10} \varphi_1)]^2; \end{aligned} \quad (14.3)$$

$$\Pi_{14} = \frac{1}{2}k_{14}[(z_2 + a_3\phi_2) - (z_3 - a_9\phi_3)]^2; k_i - \text{coefficients of stiffness of the } i\text{-element.}$$

The dissipation function of the vehicle is defined by the following relation:

$$\begin{aligned} \Phi &= \sum_{i=1}^{14} \Phi_i, \text{ where } \Phi_1 = \frac{1}{2}h_1[(\dot{z}_2 - a_4\dot{\phi}_1) - (\dot{z}_2 - (a_1 + a_4)\dot{\phi}_2)]^2; \\ \Phi_2 &= \frac{1}{2}h_2[(\dot{z}_1 + a_5\dot{\phi}_1) - (\dot{z}_2 - (a_1 - a_5)\dot{\phi}_2)]^2; \quad \Phi_3 = \frac{1}{2}h_3[\dot{z}_4 - (\dot{z}_2 - a_1\dot{\phi}_2)]^2; \\ \Phi_4 &= \frac{1}{2}h_4[\dot{z}_5 - (\dot{z}_2 - a_1\dot{\phi}_2)]^2; \quad \Phi_5 = \frac{1}{2}h_5[(\dot{z}_3 + a_6\dot{\phi}_3) - \dot{z}_6]^2; \\ \Phi_6 &= \frac{1}{2}h_6[(\dot{z}_3 + a_7\dot{\phi}_3) - \dot{z}_7]^2; \quad \Phi_7 = \frac{1}{2}h_7[(\dot{z}_3 + a_8\dot{\phi}_3) - \dot{z}_8]^2; \\ \Phi_8 &= \frac{1}{2}h_8(\dot{z}_4 - \dot{z}_8)^2; \quad \Phi_9 = \frac{1}{2}h_9(\dot{z}_5 - \dot{z}_{11})^2; \quad \Phi_{10} = \frac{1}{2}h_{10}(\dot{z}_6 - \dot{z}_{12})^2; \quad \Phi_{11} = \frac{1}{2}h_{11}(\dot{z}_7 - \dot{z}_{13})^2; \\ \Phi_{12} &= \frac{1}{2}h_{12}(\dot{z}_8 - \dot{z}_{14})^2; \quad \Phi_{13} = \frac{1}{2}h_{13}[\dot{z}_9 - (\dot{z}_1 + a_{10}\dot{\phi}_1)]^2; \\ \Phi_{14} &= \frac{1}{2}h_{14}[(\dot{z}_2 + a_3\dot{\phi}_2) - (\dot{z}_3 - a_9\dot{\phi}_3)]^2; h_i - \text{coefficient of damping of the } i\text{-element} \end{aligned} \quad (14.4)$$

For magnification of comfortability of the driver its seat is connected to the pneumocylinder. The linear stiffness of a pneumatic suspension of the seat of the driver is defined by the following relation:

$$k_B = \frac{p_{10}S_1^2\chi}{V_{10}} \left[ 1 - \frac{p_{20}V_{10}}{p_{10}V_{20}} \left( \frac{S_2}{S_1} \right)^2 \right] \quad (14.5)$$

where  $p_{10}$ ,  $p_{20}$  - pressure in chambers of the pneumocylinder ( $p_{20}$  - pressure in the data-flow chamber);  $S_1$ ,  $S_2$  - area of pneumocylinders ( $S_1 > S_2$ );  $V_{10}$ ,  $V_{20}$  - volumes of chambers of the pneumocylinder;  $\chi$  - polytropic exponent. The vector of generalized coordinates has the following form:

$$\{Z\}^T = [z_1 \ \phi_1 \ z_2 \ \phi_2 \ z_3 \ \phi_3 \ z_4 \ z_5 \ z_6 \ z_7 \ z_8 \ z_9 \ z_{10} \ z_{11} \ z_{12} \ z_{13} \ z_{14}] \quad (14.6)$$

The common equations system of motion of a vehicle accepts the next form:

$$[M]\{\ddot{Z}\} + [C]\{\dot{Z}\} + [K]\{Z\} = \{Q\} \quad (14.7)$$

The section of a road along which the vehicle moves, is divided by some number of road elements (fig. 14.2). The unevenness of a road in each such element is approximated by the next expression:

$$q(x) = N_{i-1}(x) q_{i-1} + N_i(x) q_i; \quad N_{i-1}(x) = \frac{x_i - x}{x_i - x_{i-1}}; \quad (14.8)$$

$$N_i(x) = \frac{x - x_{i-1}}{x_i - x_{i-1}}, \quad (14.9)$$

where  $N_{i-1}$ ,  $N_i$  - height of unevennesses of road in points  $i-1$  and  $i$  accordingly.

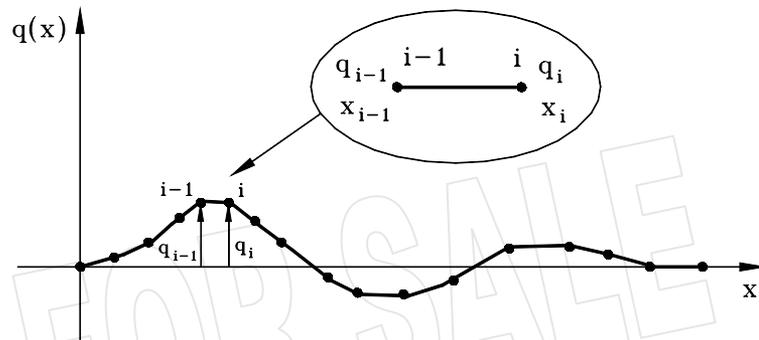


Fig. 14.2. A model of road unevennesses

The common equations system of motion are integrated by the Runge-Kutta method. The *conditions of contact* between the vehicle busbar and surface of a road are set as follows:

$$z_i = \begin{cases} z_i, & \text{when } z_i \geq q_i; \\ q_i, & \text{when } z_i < q_i; \end{cases} \quad (14.10)$$

$$\dot{z}_i = \begin{cases} \dot{z}_i, & \text{when } z_i \geq q_i \text{ и } \dot{z}_i \geq 0; \\ 0, & \text{when } z_i \leq q_i \text{ и } \dot{z}_i < 0; \end{cases} \quad (14.11)$$

$$\ddot{z}_i = \begin{cases} F_i, & \text{when } z_i \geq q_i \text{ и } F_i > 0; \\ 0, & \text{when } z_i \leq q_i \text{ и } F_i \leq 0; \end{cases} \quad (14.12)$$

where  $F_i$  – right part of the equation, resolved relative to  $\dot{z}_i$  and  $i=10..14$ .

#### 14.1.2. Input data for the solution of the problem

For solution of a problem of motion of a vehicle along a rough road the *Trailer* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable  $F$ , necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are recorded: number of points of a surface of a road ( $npoin$ ); number of degrees of freedom, for which the values of displacements, velocities and accelerations are determined ( $nwr$ ); velocity of a motion of a vehicle; initial  $x$ -coordinate of a front axis of the vehicle on a surface of the road ( $x_{10}$ ).

In the *second* line the next parameters are recorded: a print code of intermediate results ( $kprint$ ); initial pressures ( $P_{10}, P_{20}$ ); volumes of chambers ( $V_{10}, V_{20}$ ); cross-sectional areas ( $S_1, S_2$ ) pneumo-cylinders of a suspension and seat of the driver; a polytropic exponent ( $kpolit$ ). If  $kprint=0$ , then the intermediate results are not printed; otherwise, they are printed out. In the *third* line the next parameters are recorded: number indicating through what amount of integration steps the results of calculations are recorded ( $nstep$ ); number of integration steps ( $ntime$ ) and an integration step ( $dtime$ ).

In each subsequent 14 lines, the elements of the array  $Amase(14, 2)$  are coded. Into each line of the datafile the number of a solid body, its mass and moment of inertia of a solid body are recorded. Behind of the array  $Amase$  into the datafile the elements of the  $AL(10)$  array are recorded line by line. Into each line of the datafile the line number and an element of the  $AL$  array are recorded. Into each line of the  $AL$  array the geometrical parameter  $a_i$  is recorded.

Behind of the array **AL** into the datafile the elements of the arrays **Astif(14)** and **Adamp(14)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Astif** and **Adamp** are recorded. Into each line of the arrays **Astif** and **Adamp** coefficients of *stiffnesses* and *damping* of elements are recorded accordingly.

Behind of the arrays **Astif** and **Adamp** into the datafile the elements of the arrays **Xprofil(npoin)** and **Qprofil(npoin)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Xprofil** and **Qprofil** are recorded. Into each line of the arrays **Xprofil** and **Qprofil** x-coordinate and height of unevenness of a road are recorded accordingly.

Behind of the arrays **Xprofil** and **Qprofil** into the datafile the elements of the **Mwr(nwr)** array are recorded line by line. Into each line of the datafile the line number and element of the array **Mwr** are recorded. Into each line of the array **Mwr** the number of a degree of freedom is recorded, for which the values of displacements, velocities and accelerations are defined. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*npoin, nwr, veloc, x10, kprint, p10, p20, V10, V20, S1, S2, kpolit, nstep, ntime, dtime*  
Text line \*  
Array **Amase(14, 2)**  
Text line \*  
Array **AL(10)**  
Text line \*  
Arrays **Astif(14), Adamp(14)**  
Text line \*  
Arrays **Xprofil(npoin), Qprofil(npoin)**  
Text line \*  
Array **Mwr(nwr)**

### 14.1.3. Brief description of the Trailer program solving the problem

The *Trailer* program was programmed on the *Maple*-language; it consists of the basic program and 18 procedures. All procedures can be divided into three groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of integration steps of the differential equations. The program calculates displacements, velocities and accelerations of the concentrated masses of a vehicle in a time dependence of motion along a rough surface of a road, and also eigenvalues and natural frequencies. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3* and *file\_rez4* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of *displacements*, into the file *file\_rez2* values of *velocities*, into the file *file\_rez3* values of *accelerations*, at last into the file *file\_rez4* eigenvalues and natural frequencies are recorded accordingly.

### 14.1.4. An example of use of the Maple-program Trailer

As an example of application of the *Trailer* program, the motion of the vehicle along surface of a road is considered on which the *barrier* restricting velocity of a motion is installed ("*sleeping policeman*"); scheme of the barrier is represented on the *fig. 14.3*.

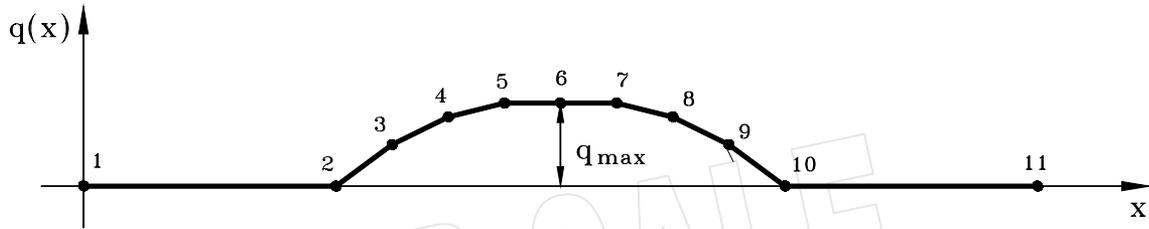


Fig. 14.3. A profile of unevennesses of a road ( $q_{\max} = 0.05$  m)

Input data for the test example: velocity of a motion  $veloc=50$  km/h; number of points of road surface  $npoin=11$ ;  $nwr=4$ ; initial x-coordinate of a front axis of the vehicle on a road surface  $x10=14.5$  m;

$kpolit=1.4$ ;  $p_{10} = 0,5$  MPa,  $p_{20} = 0,4$  MPa,  $V_{10} = 1,570 \cdot 10^{-3}$  m<sup>3</sup>,  $V_{20} = 1,9631 \cdot 10^{-4}$  m<sup>3</sup>,

$S_1 = 0,785 \cdot 10^{-2}$  m<sup>2</sup>,  $S_2 = 0,785 \cdot 10^{-2}$  m<sup>2</sup>,  $kpolit=1.4$ ;  $m_1 = 950$  kg ;  $m_2 = 3450$  kg ;

$m_3 = 24800$  kg ;  $m_4 = 803$  kg ;  $m_5 = m_6 = m_7 = m_8 = 1503$  kg ;  $m_9 = 100$  kg ;

$m_{10} = m_{11} = m_{12} = m_{13} = m_{14} = 1503$  kg ;  $I_1 = 800$  kg·m<sup>2</sup> ;  $I_2 = 9500$  kg·m<sup>2</sup> ;

$I_3 = 200000$  kg·m<sup>2</sup> ;  $I_4 = I_5 = I_6 = I_7 = I_8 = I_9 = I_{10} = I_{11} = I_{12} = I_{13} = I_{14} = 0$  ;  $a_1 = 0,7$  m ;

$a_2 = 2,8$  m ;  $a_3 = 2,2$  m ;  $a_4 = 0,85$  m ;  $a_5 = 1,15$  m ;  $a_6 = 3,0$  m ;  $a_7 = 3,68$  m ;

$a_8 = 4,36$  m ;  $a_9 = 5,62$  m ;  $a_{10} = 1,0$  m ;  $k_1 = k_2 = 8 \cdot 10^4$   $\frac{N}{m}$  ;  $k_3 = 4 \cdot 10^5$   $\frac{N}{m}$  ;

$k_5 = k_6 = k_7 = 1 \cdot 10^5$   $\frac{N}{m}$  ;  $k_8 = 2 \cdot 10^6$   $\frac{N}{m}$  ;  $k_9 = k_{10} = k_{11} = k_{12} = 4 \cdot 10^6$   $\frac{N}{m}$  ;  $k_{13} = 0$  ;

$k_{14} = 1 \cdot 10^9$   $\frac{N}{m}$  ;  $h_1 = h_2 = 0,475 \cdot 10^3$   $\frac{Ns}{m}$  ;  $h_3 = 3,3 \cdot 10^4$   $\frac{Ns}{m}$  ;  $h_4 = 5,83 \cdot 10^4$   $\frac{Ns}{m}$  ;

$h_5 = h_6 = h_7 = 8,33 \cdot 10^4$   $\frac{Ns}{m}$  ;  $h_8 = 0,7 \cdot 10^3$   $\frac{Ns}{m}$  ;  $h_9 = h_{10} = h_{11} = h_{12} = 1,2 \cdot 10^3$   $\frac{Ns}{m}$  ;  $h_{13} = 0$  ;

$h_{14} = 1,0 \cdot 10^5$   $\frac{Ns}{m}$  .

Results of calculation over the test example:

*Eigenvalues (real and imaginary part) and natural frequencies of the vehicle:*

Re[1]= -7.640335e+01	Im[1]=9.948249e+02	frequency[1]=5.019881e+00 Hz
Re[2]= -7.640335e+01	Im[2]=-9.948249e+02	frequency[2]=5.019881e+00 Hz
Re[3]= -1.842860e+01	Im[3]=3.195503e+02	frequency[3]=2.845049e+00 Hz
Re[4]= -1.842860e+01	Im[4]=-3.195503e+02	frequency[4]=2.845049e+00 Hz
Re[5]= -3.065425e+01	Im[5]=4.491190e+02	frequency[5]=3.372879e+00 Hz
Re[6]= -3.065425e+01	Im[6]=-4.491190e+02	frequency[6]=3.372879e+00 Hz
Re[7]= -3.075138e+01	Im[7]=4.490638e+02	frequency[7]=3.372672e+00 Hz
Re[8]= -3.075138e+01	Im[8]=-4.490638e+02	frequency[8]=3.372672e+00 Hz
Re[9]= -3.075792e+01	Im[9]=4.490837e+02	frequency[9]=3.372747e+00 Hz
Re[10]= -3.075792e+01	Im[10]=-4.490837e+02	frequency[10]=3.372747e+00 Hz
Re[11]= -8.094194e+01	Im[11]=0e-01	frequency[11]=0e-01 Hz
Re[12]= -5.245800e-01	Im[12]=1.745380e+01	frequency[12]=6.649136e-01 Hz
Re[13]= -5.245800e-01	Im[13]=-1.745380e+01	frequency[13]=6.649136e-01 Hz
Re[14]= -6.266947e-01	Im[14]=1.371579e+01	frequency[14]=5.894278e-01 Hz

$\text{Re}[15] = -6.266947e-01$        $\text{Im}[15] = -1.371579e+01$        $\text{frequency}[15] = 5.894278e-01$  Hz  
 $\text{Re}[16] = -1.653290e-01$        $\text{Im}[16] = 1.110861e+01$        $\text{frequency}[16] = 5.304568e-01$  Hz  
 $\text{Re}[17] = -1.653290e-01$        $\text{Im}[17] = -1.110861e+01$        $\text{frequency}[17] = 5.304568e-01$  Hz

On the *fig. 14.4* the dependences of *displacements*  $z_9$ ,  $z_1$ ,  $z_2$  and  $z_3$  of masses of the vehicle in a time, while on the *fig. 14.5* the dependences of *accelerations*  $\ddot{z}_9$ ,  $\ddot{z}_1$ ,  $\ddot{z}_2$  and  $\ddot{z}_3$  of masses of the vehicle in a time are represented.

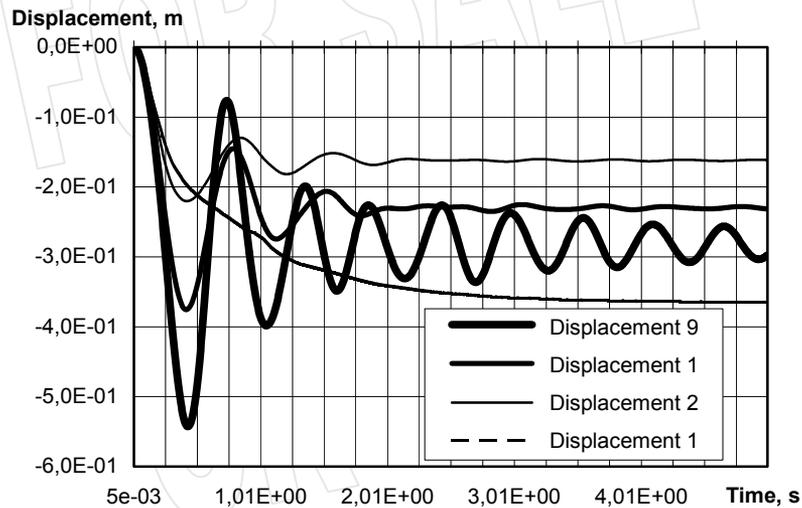


Fig. 14.4. The dependences of displacements of masses of vehicle in a time

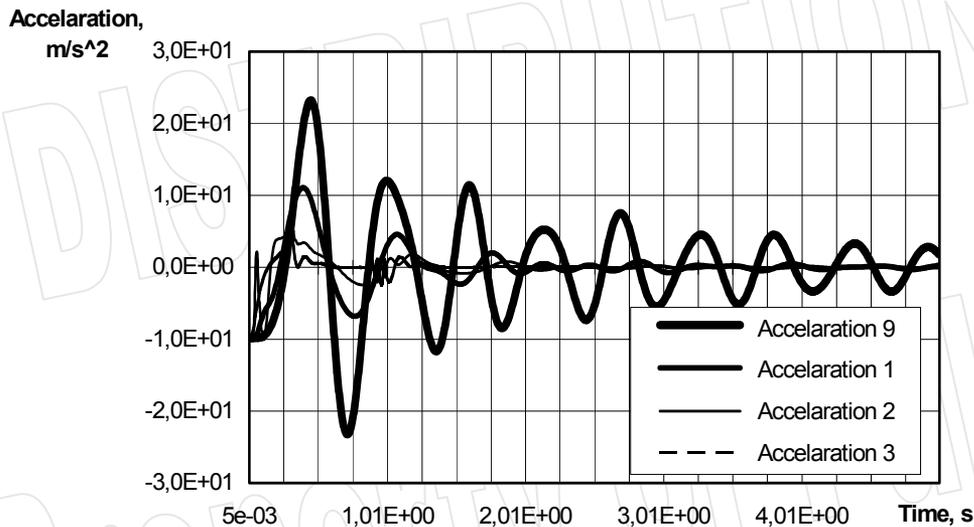


Fig. 14.5. The dependences of accelerations of masses of vehicle in a time

The *Trailer* program is intended for the solution of a problem of motion of a vehicle along a rough road. The source module of the program in *Maple-language*, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles *Mb11\_1\_new.mws*, *Mb11\_1.dat* and *Mb11\_1\_1.rez* accordingly.

## 14.2. Stationary random oscillations of a vehicle

The motion of a vehicle along a rough road surface has a random character. On various sites of a road there are unevennesses of the diversified form and extensions. The sequence of salients and troughs is purely random, therefore quantity and duration of acting of impulses of forces at a motion of the vehicle along the mentioned unevennesses is also random. The statistical analysis of results of metering of microprofiles of a road allows to receive the probability characteristics of unevennesses, namely: *average values* and *correlation functions*. The practical interest represents a receiving of the characteristics of stationary random oscillations of a vehicle at its motion with a stationary velocity along a rough road.

### 14.2.1. The calculated expressions for determination of motion of a vehicle

The *vertical oscillations* of the vehicle represented on the *fig. 14.6* are considered.

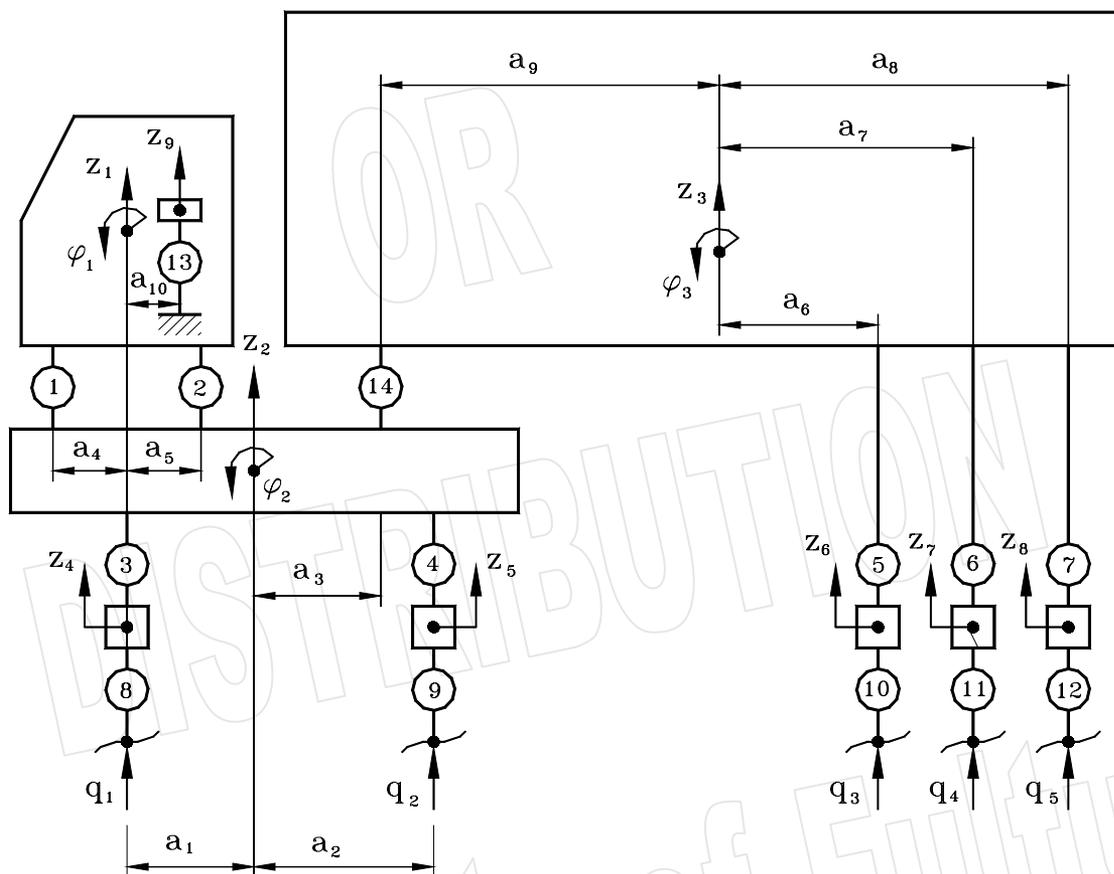


Fig. 14.6. The calculated scheme of a vehicle (by circles the elements, consisting of a spring and a dampfer, are marked)

The equations of motion of the vehicle in a vertical plane are deduced similarly to equations represented in the problem 14.1, and have the following form:

$$[\mathbf{M}]\{\ddot{\mathbf{z}}\} + [\mathbf{C}]\{\dot{\mathbf{z}}\} + [\mathbf{K}]\{\mathbf{z}\} = [\mathbf{B}_0]\{\mathbf{q}\} + [\mathbf{B}_1]\{\dot{\mathbf{q}}\}, \quad (14.13)$$

where  $[\mathbf{M}]$ ,  $[\mathbf{C}]$ ,  $[\mathbf{K}]$  - matrix of masses, of damping and stiffness, accordingly;  $\{\mathbf{z}\}$  - vector of the generalized coordinates and  $\{\mathbf{z}\}^T = [z_1 \ \varphi_1 \ z_2 \ \varphi_2 \ \varphi_3 \ z_4 \ z_5 \ z_6 \ z_7 \ z_8 \ z_9]$ ;

$$\begin{aligned}
 [B_0] = & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ k_8 & 0 & 0 & 0 & 0 \\ 0 & k_9 & 0 & 0 & 0 \\ 0 & 0 & k_{10} & 0 & 0 \\ 0 & 0 & 0 & k_{11} & 0 \\ 0 & 0 & 0 & 0 & k_{12} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}; \quad [B_1] = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ h_8 & 0 & 0 & 0 & 0 \\ 0 & h_9 & 0 & 0 & 0 \\ 0 & 0 & h_{10} & 0 & 0 \\ 0 & 0 & 0 & h_{11} & 0 \\ 0 & 0 & 0 & 0 & h_{12} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}; \quad \{q\} - \text{vector of kinematic} \\
 & \text{excitations; } \{q\}^T = [q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5]; \quad \{\dot{q}\}^T = [\dot{q}_1 \quad \dot{q}_2 \quad \dot{q}_3 \quad \dot{q}_4 \quad \dot{q}_5].
 \end{aligned}$$

The probability characteristics of components of the vector  $\{q\}$  are known; in particular, the *spectral densities*  $S_{q_i}(\omega)$  and *mutual spectral densities*  $S_{q_i q_j}(\omega)$  are known also. The *stationary solution* of the equations system (14.13) is represented as follows:

$$\{Z\} = \int_{-\infty}^{\infty} \{Z_0\} e^{i\omega t} d\omega \tag{14.14}$$

while the *vector of kinematic perturbations* is represented as Fourier integral:

$$\{q\} = \int_{-\infty}^{\infty} \{Q\} e^{i\omega t} d\omega \tag{14.15}$$

Then the *solution* of the equations system (14.14) accepts the following form:

$$\{Z_0\} = [H]\{Q\}, \text{ where: } [H] = [-\omega^2 M + i\omega C + K]^{-1} [B_0 + i\omega B_1] \tag{14.16}$$

The case is considered, when kinematic perturbations  $q_k(t)$  are expressed through a random perturbation  $q_1(t)$  and retardation as follows:

$$q_i(t) = q_1(t - t_i), \quad i=2...5, \text{ where: } t_i - \text{time of retardation} \tag{14.17}$$

The *matrix of spectral densities* of kinematic perturbations has the next form:

$$[S_q] = [S_{q_i}] \begin{bmatrix} 1 & e^{-i\omega\tau_{12}} & e^{-i\omega\tau_{13}} & e^{-i\omega\tau_{14}} & e^{-i\omega\tau_{15}} \\ e^{i\omega\tau_{12}} & 1 & e^{-i\omega\tau_{23}} & e^{-i\omega\tau_{24}} & e^{-i\omega\tau_{25}} \\ e^{i\omega\tau_{13}} & e^{i\omega\tau_{23}} & 1 & e^{-i\omega\tau_{34}} & e^{-i\omega\tau_{35}} \\ e^{i\omega\tau_{14}} & e^{i\omega\tau_{24}} & e^{i\omega\tau_{34}} & 1 & e^{-i\omega\tau_{45}} \\ e^{i\omega\tau_{15}} & e^{i\omega\tau_{25}} & e^{i\omega\tau_{35}} & e^{i\omega\tau_{45}} & 1 \end{bmatrix} \tag{14.18}$$

where  $S_{q_i}$  – spectral density of kinematic excitation acting onto a front axis of the vehicle;  $\tau_{ij} = \frac{L_{ij}}{v}$ ,

$i \neq j = 1, 2, 3, 4, 5$ ;  $L_{ij}$  - distance between wheels. The *matrix of spectral densities* of the solutions accepts the following form:

$$[S_z] = [H(i\omega)] [S_q] [\bar{H}(i\omega)]^T, \text{ where } [\bar{H}(i\omega)] - \text{adjoint matrix} \quad (14.19)$$

The *spectral density* of kinematic excitation acting onto a front axis of the vehicle, can be represented as follows:

$$S_{q_1} = \frac{\sum_{i=0}^{ns_1} \sum_{j=0}^{ns_2} a_{ij} \omega^i v^j}{\sum_{i=0}^{nv_1} \sum_{j=0}^{nv_2} b_{ij} \omega^i v^j} \quad (14.20)$$

where:  $\omega$  - frequency;  $v$  - velocity of the vehicle motion. The *variances* of the generalized coordinates and their *derivatives* are defined as follows:

$$[D_z] = \frac{1}{2\pi} \int_{-\infty}^{\infty} \text{diag}[S_z] d\omega; [D_{\dot{z}}] = \frac{1}{2\pi} \int_{-\infty}^{\infty} \omega^2 \text{diag}[S_z] d\omega; [D_{\ddot{z}}] = \frac{1}{2\pi} \int_{-\infty}^{\infty} \omega^4 \text{diag}[S_z] d\omega \quad (14.21)$$

### 14.2.2. Input data for the solution of the problem

For solution of a problem of definition of random oscillations of a vehicle, moving along a rough road the *Trailer\_random* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable **F**, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the numbers of the terms, whose numerator and denominator are not equal zero, for expression of a spectral density of kinematic excitation of a road are coded (*parameters nls, nlv*).

In the *second* line the next parameters are recorded: a print code of intermediate results (*kprint*); initial pressures (**p10, p20**); volumes of chambers (**V10, V10**); cross-sectional areas (**S1, S2**) pneumocylinders of a suspension and seat of the driver; a polytropic exponent (*kpolit*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. In the *third* line the next parameters are recorded: number of degrees of freedom, for which the characteristics of random oscillations are defined (*nwr*); number of points of velocity of motion of the vehicle, for which the values of the characteristics of random quantities are defined (*nv*); the minimal velocity of motion of the vehicle (*vmin*) and a step of change of velocity of motion (*dv*).

In each subsequent 9 lines, the elements of the array **Amase(9, 2)** are coded. Into each line of the datafile the number of a solid body, its mass and moment of inertia of a solid body are recorded. Behind of the array **Amase** into the datafile the elements of the **AL(10)** array are recorded line by line. Into each line of the datafile the line number and an element of the **AL** array are recorded. Into each line of the **AL** array the geometrical parameter  $a_i$  is recorded.

Behind of the array **AL** into the datafile the elements of the arrays **Astif(14)** and **Adamp(14)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Astif** and **Adamp** are recorded. Into each line of the arrays **Astif** and **Adamp** coefficients of stiffnesses and damping of elements are recorded accordingly.

Behind of the arrays **Astif** and **Adamp** into the datafile the elements of the arrays **Mnls**(*nls*, 2) and **As**(*nls*) are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Mnls** and **As** are recorded. Into each line of the arrays **Mnls** and **As** the exponents for frequency and velocity, and also value of coefficient, which are in numerator of expression of a spectral density of kinematic excitation of a road are recorded.

Behind of the arrays **Mnls** and **As** into the datafile the elements of the arrays **Mnlv**(*nlv*, 2) and **Av**(*nlv*) are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Mnlv** and **Av** are recorded. Into each line of the arrays **Mnlv** and **Av** the exponents for frequency and velocity, and also value of coefficient, which are in denominator of expression of a spectral density of kinematic excitation of a road are recorded.

Behind of the arrays **Mnlv** and **Av** into the datafile the elements of the **Mwr**(*nwr*) array are recorded line by line. Into each line of the datafile the line number and element of the array **Mwr** are recorded. Into each line of the array **Mwr** the number of a degree of freedom is recorded, for which the characteristics of random oscillations are defined. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*nls, nlv, x10, kprint, p10, p20, V10, V20, S1, S2, kpolit, nwr, nv, vmin, dv*  
Text line \*  
Array **Amase**(9, 2)  
Text line \*  
Array **AL**(10)  
Text line \*  
Arrays **Astif**(14), **Adamp**(14)  
Text line \*  
Arrays **Mnls**(*nls*, 2), **As**(*nls*)  
Text line \*  
Arrays **Mnlv**(*nlv*, 2), **Av**(*nlv*)  
Text line \*  
Array **Mwr**(*nwr*)

### 14.2.3. Brief description of the Trailer\_random program solving the problem

The *Trailer\_random* program was programmed on the *Maple*-language; it consists of the basic program and 17 procedures. All procedures can be divided into three groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of points of velocity of motion of a vehicle, for which the characteristics of random oscillations are determined. The program calculates the mean square deviations of displacements, velocities and accelerations of the concentrated masses of a vehicle in a time dependence of motion along a rough surface of a road, and also eigenvalues and oscillations frequencies of the vehicle. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3* and *file\_rez4* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of mean square deviation of *displacements*, into the file *file\_rez2* values of mean square deviation of *velocities*, into the file *file\_rez3* values of mean square deviation of accelerations, at last into the file *file\_rez4* *eigenvalues* and *oscillations frequencies* of the vehicle are recorded accordingly.

#### 14.2.4. An example of use of the *Maple*-program *Trailer\_random*

As an example of application of the *Trailer\_random* program, the motion of the vehicle represented on the *fig. 14.6*, along a road surface is considered; at that the spectral density of the road is defined

by the following relation: 
$$S_{q_i} = \frac{183,21 \omega^4 v - 545,2 \omega^2 v^3 + 413,2 v^5}{\omega^6 + 9,004 \omega^4 v^2 - 38,15 \omega^2 v^3 + 27,17 v^6}.$$

*Input data for the test example:*  $nls=3, nlv=4, kpolit=1.4, nwr=3, p_{10}=0,5 \text{ MPa}, p_{20}=0,4 \text{ MPa},$

$$V_{10} = 1,570 \cdot 10^{-3} \text{ m}^3, V_{20} = 1,9631 \cdot 10^{-4} \text{ m}^3, S_1 = 0,785 \cdot 10^{-2} \text{ m}^2, S_2 = 0,785 \cdot 10^{-2} \text{ m}^2,$$

$$nv=11, dv=2.7 \text{ m/s}; m_1 = 950 \text{ kg}; m_2 = 3450 \text{ kg}; m_3 = 24800 \text{ kg}; m_4 = 803 \text{ kg};$$

$$m_5 = m_6 = m_7 = m_8 = 1503 \text{ kg}; m_9 = 100 \text{ kg}; l_1 = 800 \text{ kg} \cdot \text{m}^2; l_2 = 9500 \text{ kg} \cdot \text{m}^2;$$

$$l_3 = 200000 \text{ kg} \cdot \text{m}^2; l_4 = l_5 = l_6 = l_7 = l_8 = l_9 = 0; a_1 = 0,7 \text{ m}; a_2 = 2,8 \text{ m}; a_3 = 2,2 \text{ m};$$

$$a_4 = 0,85 \text{ m}; a_5 = 1,15 \text{ m}; a_6 = 3,0 \text{ m}; a_7 = 3,68 \text{ m}; a_8 = 4,36 \text{ m}; a_9 = 5,62 \text{ m};$$

$$k_1 = k_2 = 8 \cdot 10^4 \frac{\text{N}}{\text{m}}; k_3 = 4 \cdot 10^5 \frac{\text{N}}{\text{m}}; k_4 = 7 \cdot 10^5 \frac{\text{N}}{\text{m}}; k_5 = k_6 = k_7 = 1 \cdot 10^5 \frac{\text{N}}{\text{m}};$$

$$k_8 = 2 \cdot 10^6 \frac{\text{N}}{\text{m}}; k_9 = k_{10} = k_{11} = k_{12} = 4 \cdot 10^6 \frac{\text{N}}{\text{m}}; k_{13} = 0; k_{14} = 1 \cdot 10^9 \frac{\text{N}}{\text{m}};$$

$$h_1 = h_2 = 0,475 \cdot 10^3 \frac{\text{Ns}}{\text{m}}; h_3 = 3,3 \cdot 10^4 \frac{\text{Ns}}{\text{m}}; h_4 = 5,83 \cdot 10^4 \frac{\text{Ns}}{\text{m}}; h_5 = h_6 = h_7 = 8,33 \cdot 10^4 \frac{\text{Ns}}{\text{m}};$$

$$h_8 = 0,7 \cdot 10^3 \frac{\text{Ns}}{\text{m}}; h_9 = h_{10} = h_{11} = h_{12} = 1,2 \cdot 10^3 \frac{\text{Ns}}{\text{m}}; h_{13} = 0; h_{14} = 1,0 \cdot 10^5 \frac{\text{Ns}}{\text{m}}.$$

*Results of calculation over the test example:*

On the *fig. 14.7* the *spectral density of kinematic excitation*, while on the *fig. 14.8* the dependences of a mean square deviation of accelerations  $\ddot{z}_9, \ddot{z}_1, \ddot{z}_2, \ddot{z}_3$  from velocity of motion of a vehicle are represented.

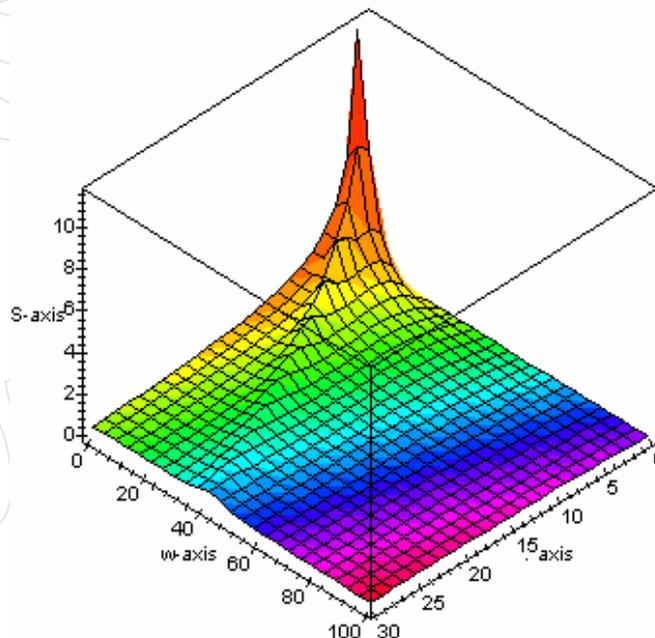


Fig. 14.7. Kinematic density of spectral excitation

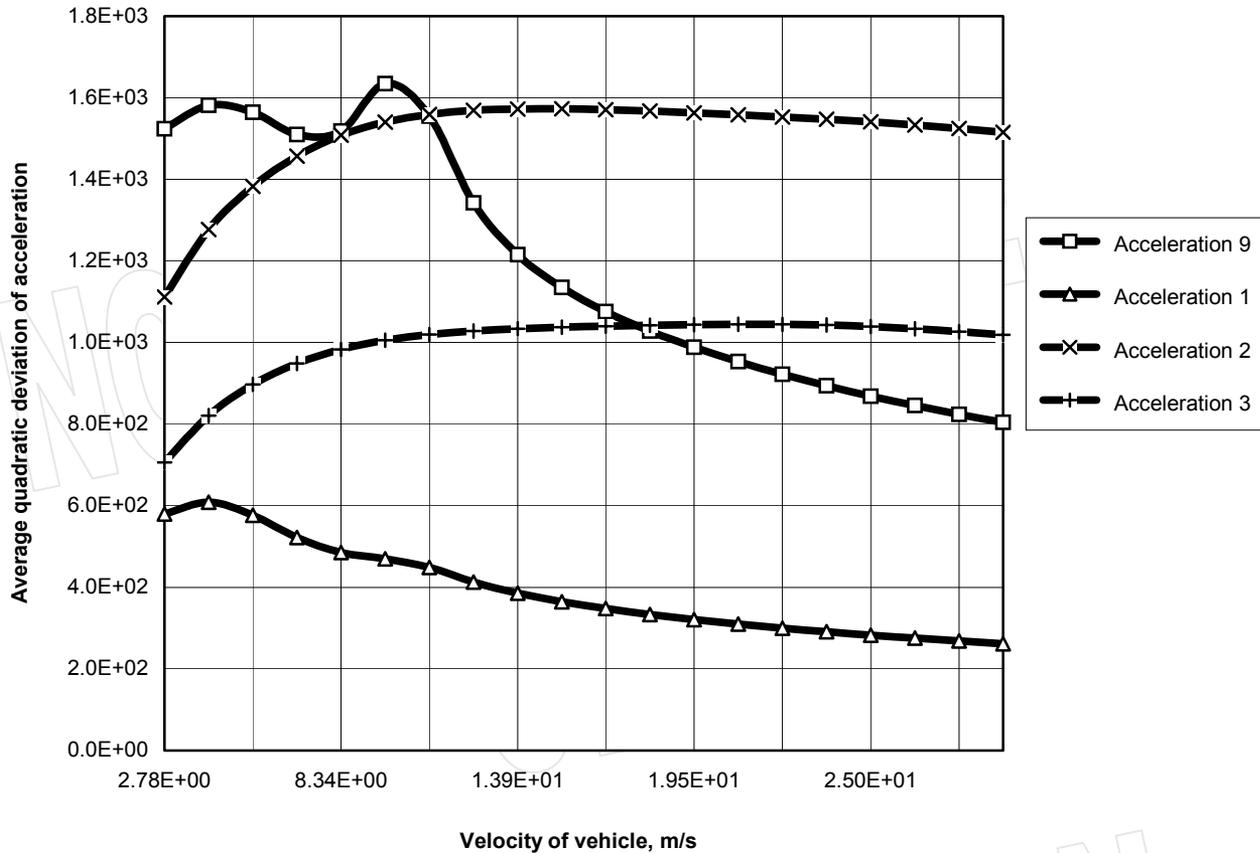


Fig. 14.8. Dependences of a mean square deviation of accelerations from velocity of motion of a vehicle

The *Trailer\_random* program is intended for the solution of a problem on definition of random oscillations of a vehicle moving along a rough road. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb11\_2\_new.mws*, *Mb11\_2.dat* and *Mb11\_2\_1.rez* accordingly.

### 14.3. Motion of a carriage along a rough railway

Safety of motion of trains, and also rhythm and profitability of a work of a railway transport essentially depend on a construction and state of a railway and rolling-stock. A railway and a rolling-stock, in particular a carriage, represent a united mechanical system. In actual conditions the rails and wheels have unevennesses on surface of a rolling, and also some other technical features, therefore in elements of a railway and rolling-stock arise various oscillations. In the represented problem the oscillations of a carriage in vertical plane at its motion along a rough railway are considered.

#### 14.3.1. The calculated expressions for determination of a carriage motion

A carriage motion in a vertical plane is considered. The calculated scheme of the carriage consisting of a body, bogies chassis with axle-boxes suspension, wheel pairs, is represented on the *fig. 14.9*.

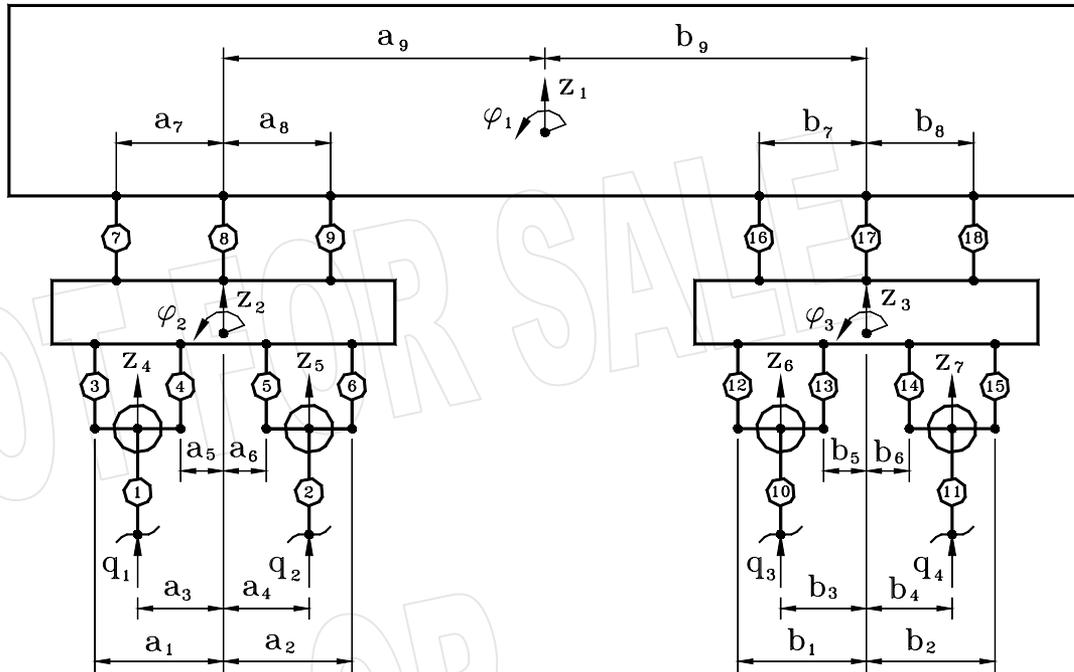


Fig. 14.9. The calculated scheme of a carriage (by circles the elements, consisting of a spring and a damper, are marked)

The equations of a carriage motion are deduced on the basis of the Lagrange equation of the second type:

$$\frac{d}{dt} \left( \frac{\partial T}{\partial \{\dot{q}_i\}} \right) - \frac{\partial T}{\partial \{q_i\}} + \frac{\partial \Pi}{\partial \{q_i\}} + \frac{\partial \Phi}{\partial \{\dot{q}_i\}} = \{Q_i\} \quad (14.22)$$

where  $T$ ,  $\Pi$  - kinetic and potential energy of a carriage;  $\Phi$  - dissipation function;  $q_i$ ,  $\dot{q}_i$  - vectors of the generalized coordinates and velocities;  $\{Q\}$  - vector of the generalized forces;  $t$  - time. Then the kinetic energy of the carriage is defined by the following relation:

$$T = \frac{1}{2} \sum_{i=1}^3 (m_i \dot{z}_i^2 + I_i \dot{\phi}_i^2) + \frac{1}{2} \sum_{i=4}^7 m_i \dot{z}_i^2 \quad (14.23)$$

where  $m_i$ ,  $I_i$  - mass and moment of inertia of mass of the  $i$ -th body;  $\dot{z}_i$ ,  $\dot{\phi}_i$  - the generalized velocities of the  $i$ -th body. The potential energy of the carriage is defined by the following relation:

$$\begin{aligned} \Pi &= \sum_{i=1}^{18} \Pi_i, \text{ where } \Pi_1 = \frac{1}{2} k_1 (z_4 - q_1)^2; \Pi_2 = \frac{1}{2} k_2 (z_5 - q_2)^2; \Pi_3 = \frac{1}{2} k_3 [(z_2 - a_1 \phi_2) - z_4]^2; \\ \Pi_4 &= \frac{1}{2} k_4 [(z_2 - a_5 \phi_2) - z_4]^2; \Pi_5 = \frac{1}{2} k_5 [(z_2 + a_6 \phi_2) - z_5]^2; \Pi_6 = \frac{1}{2} k_6 [(z_2 + a_2 \phi_2) - z_5]^2; \\ \Pi_7 &= \frac{1}{2} k_7 [(z_1 - (a_7 + a_9) \phi_1) - (z_2 - a_7 \phi_2)]^2; \Pi_8 = \frac{1}{2} k_8 [(z_1 - a_9 \phi_1) - z_2]^2; \\ \Pi_9 &= \frac{1}{2} k_9 [(z_1 - (a_9 - a_8) \phi_1) - (z_2 + a_8 \phi_2)]^2; \Pi_{10} = \frac{1}{2} k_{10} (z_6 - q_3)^2; \end{aligned} \quad (14.24)$$

$$\begin{aligned}\Pi_{11} &= \frac{1}{2}k_{11}(z_7 - q_4)^2; \Pi_{12} = \frac{1}{2}k_{12}[(z_3 - b_1\phi_3) - z_6]^2; \Pi_{13} = \frac{1}{2}k_{13}[(z_3 - b_5\phi_3) - z_6]^2; \\ \Pi_{14} &= \frac{1}{2}k_{14}[(z_3 + b_6\phi_3) - z_7]^2; \Pi_{15} = \frac{1}{2}k_{15}[(z_3 + b_2\phi_3) - z_7]^2; \\ \Pi_{16} &= \frac{1}{2}k_{16}[(z_1 + (b_9 - b_7)\phi_1) - (z_3 - b_7\phi_3)]^2; \Pi_{17} = \frac{1}{2}k_{17}[(z_1 + b_9\phi_1) - z_3]^2; \\ \Pi_{18} &= \frac{1}{2}k_{18}[(z_1 + (b_9 + b_8)\phi_1) - z_3]^2;\end{aligned}$$

$k_i$  - coefficients of stiffness of the  $i$ -element. The *dissipation function* of the vehicle is defined by the following relation:

$$\begin{aligned}\Phi &= \sum_{i=1}^{18} \Phi_i, \text{ where } \Phi_1 = \frac{1}{2}h_1(\dot{z}_4 - \dot{q}_1)^2; \Phi_2 = \frac{1}{2}h_2(\dot{z}_5 - \dot{q}_2)^2; \Phi_3 = \frac{1}{2}h_3[(\dot{z}_2 - a_1\dot{\phi}_2) - \dot{z}_4]^2; \\ \Phi_4 &= \frac{1}{2}h_4[(\dot{z}_2 - a_5\dot{\phi}_2) - \dot{z}_4]^2; \Phi_5 = \frac{1}{2}h_5[(\dot{z}_2 + a_6\dot{\phi}_2) - \dot{z}_5]^2; \Phi_6 = \frac{1}{2}h_6[(\dot{z}_2 + a_2\dot{\phi}_2) - \dot{z}_5]^2; \\ \Phi_7 &= \frac{1}{2}h_7[(\dot{z}_1 - (a_7 + a_9)\dot{\phi}_1) - (\dot{z}_2 - a_7\dot{\phi}_2)]^2; \Phi_8 = \frac{1}{2}h_8[(\dot{z}_1 - a_9\dot{\phi}_1) - \dot{z}_2]^2; \\ \Phi_9 &= \frac{1}{2}h_9[(\dot{z}_1 - (a_9 - a_8)\dot{\phi}_1) - (\dot{z}_2 + a_8\dot{\phi}_2)]^2; \Phi_{10} = \frac{1}{2}h_{10}(\dot{z}_6 - \dot{q}_3)^2; \quad (14.25) \\ \Phi_{11} &= \frac{1}{2}h_{11}(\dot{z}_7 - \dot{q}_4)^2; \Phi_{12} = \frac{1}{2}h_{12}[(\dot{z}_3 - b_1\dot{\phi}_3) - \dot{z}_6]^2; \Phi_{13} = \frac{1}{2}h_{13}[(\dot{z}_3 - b_5\dot{\phi}_3) - \dot{z}_6]^2; \\ \Phi_{14} &= \frac{1}{2}h_{14}[(\dot{z}_3 + b_6\dot{\phi}_3) - \dot{z}_7]^2; \Phi_{15} = \frac{1}{2}h_{15}[(\dot{z}_3 + b_2\dot{\phi}_3) - \dot{z}_7]^2; \\ \Phi_{16} &= \frac{1}{2}h_{16}[(\dot{z}_1 + (b_9 - b_7)\dot{\phi}_1) - (\dot{z}_3 - b_7\dot{\phi}_3)]^2; \Phi_{17} = \frac{1}{2}h_{17}[(\dot{z}_1 + b_9\dot{\phi}_1) - \dot{z}_3]^2; \\ \Phi_{18} &= \frac{1}{2}h_{18}[(\dot{z}_1 + (b_9 + b_8)\dot{\phi}_1) - \dot{z}_3]^2;\end{aligned}$$

$h_i$  - coefficient of damping of the  $i$ -element. The trajectory of a wheel motion on railway joints a can be approximated by the next expression [107]:

$$q_1 = \left| a_1 \sin\left(\frac{1}{2}\omega t\right) + a_2 \sin\left(\frac{3}{2}\omega t\right) \right|, \quad (14.26)$$

where  $a_1, a_2$  - amplitudes of unevennesses;  $\omega$  - frequency of affecting of railway joints onto a carriage moving with velocity  $v$ ,  $\omega = \frac{2\pi}{L_r} v$ ;  $L_r$  - length of a rail.

### 14.3.2. Input data for the solution of the problem

For solution of a problem of motion of a carriage along a rough railway the *Carriage* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable  $F$ , necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are recorded: a print code of intermediate results (*parameter kprint*); velocity of a carriage motion (*veloc*); an initial coordinate of a front axis of wheels of carriage ( $x_{10}$ ); length of a rail (*alp*); amplitudes of the *first* and the *second* harmonics

(*amp1*, *amp2*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. In the *second* line the next parameters are recorded: number of degrees of freedom, for which into the resultant file the values of unknowns are recorded (*nwr*), number indicating through what amount of integration steps the results of calculations are recorded (*nstep*); number of integration steps (*ntime*) and an integration step (*dttime*).

In each subsequent 7 lines, the elements of the array **Amase(7, 2)** are coded. Into each line of the datafile the number of a solid body, its mass and moment of inertia of a solid body are recorded. Behind of the array **Amase** into the datafile the elements of the arrays **AL(9)** and **BL(9)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **AL** and **BL** are recorded. Into each line of the arrays **AL** and **BL** the two geometrical parameters  $\mathbf{a}_i$  and  $\mathbf{b}_i$  are recorded accordingly.

Behind of the arrays **AL** and **BL** into the datafile the elements of the arrays **Astif(18)** and **Adamp(18)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Astif** and **Adamp** are recorded. Into each line of the arrays **Astif** and **Adamp** coefficients of stiffnesses and damping of elements are recorded accordingly. Behind of the arrays **Astif** and **Adamp** into the datafile the elements of the **Mwr(nwr)** array are recorded line by line. Into each line of the datafile the line number and element of the array **Mwr** are recorded. Into each line of the array **Mwr** the number of a degree of freedom is recorded, for which into the resultant file the values of unknowns are recorded. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

Schematic structure of the datafile:

Text line \*  
*kprint, veloc, x10, alp, amp1, amp2, nwr, nstep, ntime, dttime*  
 Text line \*  
 Array **Amase(7, 2)**  
 Text line \*  
 Arrays **AL(9), BL(9)**  
 Text line \*  
 Arrays **Astif(18), Adamp(18)**  
 Text line \*  
 Array **Mwr(nwr)**

### 14.3.3. Brief description of the Carriage program solving the problem

The *Carriage* program had been programmed on the *Maple*-language; it consists of the basic program and 17 procedures. All procedures can be divided into three groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of integration steps of the differential equations. The program calculates displacements, velocities and accelerations of the concentrated masses of a carriage, and also eigenvalues and oscillations frequencies of a carriage. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3* and *file\_rez4* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of *displacements*, into the file *file\_rez2* values of velocities, into the file *file\_rez3* values of accelerations in a time dependence, at last into the file *file\_rez4* eigenvalues and oscillations frequencies of a carriage are recorded accordingly.

### 14.3.4. An example of use of the *Maple*-program Carriage

As an example of application of the *Carriage* program, the carriage motion along a rough railway is considered (fig. 14.9).

Input data for the test example:  $nwr=2$ ,  $nstep=10$ ,  $ntime=5000$ ,  $dtime=5 \cdot 10^{-3}$  s,  $amp1=20 \cdot 10^{-3}$  m,  $amp2=10 \cdot 10^{-3}$  m,  $veloc=120$  km/h,  $x10=20$  m,  $alp=12.5$  m,  $m_1 = 45000$  kg ;  $l_1 = 90000$  kg · m<sup>2</sup>  
 $m_2 = m_3 = 2500$  kg ;  $m_4 = m_5 = m_6 = m_7 = 2000$  kg ;  $l_2 = l_3 = 3000$  kg · m<sup>2</sup>;  $a_1 = 0,7$  m  
 $l_4 = l_5 = l_6 = l_7 = 0$  ;  $a_2 = 2,8$  m ;  $a_3 = 2,2$  m ;  $a_4 = 0,85$  m ;  $a_5 = 1,15$  m ;  $a_6 = 3,0$  m ;  
 $a_7 = 3,68$  m ;  $a_8 = 4,36$  m ;  $a_9 = 5,62$  m ;  $a_{10} = 1,0$  m ;  $k_1 = k_2 = k_{10} = k_{11} = 1 \cdot 10^{11} \frac{N}{m}$  ;  
 $k_3 = k_4 = k_5 = k_6 = k_{12} = k_{13} = k_{14} = k_{15} = 2 \cdot 10^6 \frac{N}{m}$  ;  $k_7 = k_9 = k_{16} = k_{18} = 1,238 \cdot 10^5 \frac{N}{m}$  ;  
 $k_8 = k_{17} = 6,40 \cdot 10^5 \frac{N}{m}$  ;  $h_7 = h_9 = h_{16} = h_{18} = 4,76 \cdot 10^4 \frac{Ns}{m}$  ;

the remaining coefficients of a damping are supposed by zero.

#### Results of calculation over the test example:

*Eigenvalues (real and imaginary part) and natural frequencies of the carriage:*

Re[1]= -5.464747e+02	Im[1]=0e-01	frequency[1]=0e-01 Hz
Re[2]= -4.883556e+02	Im[2]=0e-01	frequency[2]=0e-01 Hz
Re[3]= -2.152341e+01	Im[3]=5.384919e+01	frequency[3]=1.167910e+00 Hz
Re[4]= -2.152341e+01	Im[4]=-5.384919e+01	frequency[4]=1.167910e+00 Hz
Re[5]= -7.474423e+00	Im[5]=4.609960e+01	frequency[5]=1.080609e+00 Hz
Re[6]= -7.474423e+00	Im[6]=-4.609960e+01	frequency[6]=1.080609e+00 Hz
Re[7]= -4.307889e+01	Im[7]=3.573762e+01	frequency[7]=9.514434e-01 Hz
Re[8]= -4.307889e+01	Im[8]=-3.573762e+01	frequency[8]=9.514434e-01 Hz
Re[9]= -1.736626e+00	Im[9]=5.759756e+00	frequency[9]=3.819637e-01 Hz
Re[10]= -1.736626e+00	Im[10]=-5.759756e+00	frequency[10]=3.819637e-01 Hz
Re[11]= -6.862419e+00	Im[11]=0e-01	frequency[11]=0e-01 Hz
Re[12]= -2.356040e+01	Im[12]=0e-01	frequency[12]=0e-01 Hz
Re[13]= -2.54818e-06	Im[13]=7.071209e+03	frequency[13]=1.338341e+01 Hz
Re[14]= -2.54818e-06	Im[14]=-7.071209e+03	frequency[14]=1.338341e+01 Hz
Re[15]= -2.550965e-06	Im[15]=7.071209e+03	frequency[15]=1.338341e+01 Hz
Re[16]= -2.550965e-06	Im[16]=-7.071209e+03	frequency[16]=1.338341e+01 Hz
Re[17]= -6.4971e-08	Im[17]=7.071209e+03	frequency[17]=1.338341e+01 Hz
Re[18]= -6.4971e-08	Im[18]=-7.071209e+03	frequency[18]=1.338341e+01 Hz
Re[19]= -6.499e-08	Im[19]=7.071209e+03	frequency[19]=1.338341e+01 Hz
Re[20]= -6.499e-08	Im[20]=-7.071209e+03	frequency[20]=1.338341e+01 Hz

On the fig. 14.10 the dependences of *displacement* and *angle of rotation* of a carriage body in a time, while on the fig. 14.11 the dependences of *linear* and *angular accelerations* of carriage body in a time are represented.

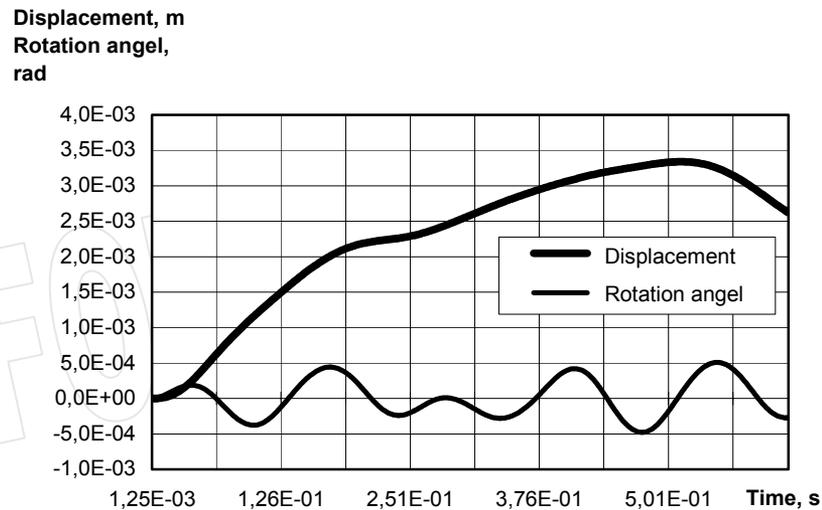


Fig. 14.10. Dependences of displacement and angle of rotation of carriage body in a time at motion velocity  $v=120$  km/h

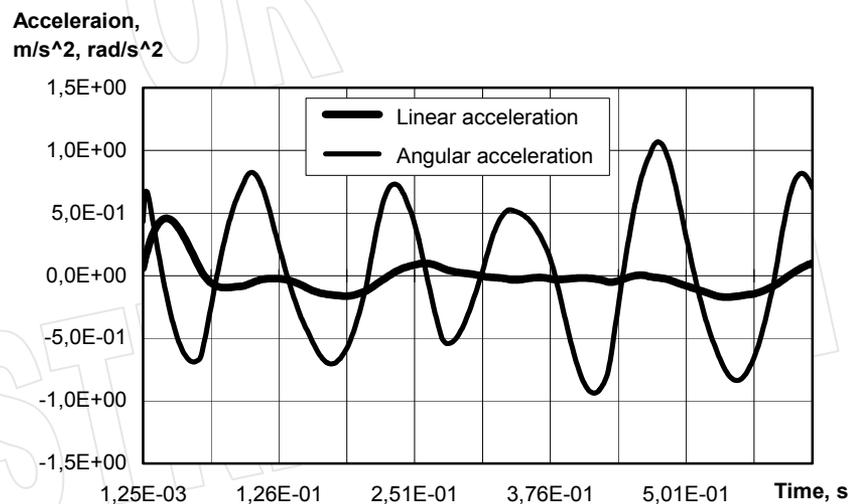


Fig. 14.11. Dependences of linear and angular accelerations of carriage body in a time at motion velocity  $v=120$  km/h

The *Carriage* program is intended for the solution of a problem of motion of a carriage along a rough railway. The source module of the program in Maple-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the present book in datafiles Mb11\_3\_new.mws, Mb11\_3.dat and Mb11\_3\_1.rez accordingly.

#### 14.4. Dynamics of pneumatic vibration-extinguisher with stationary magnets

Incremental velocities and the dynamic loadings in the modern mechanical engineering condition a increasing demands to protection of mechanisms against vibration and shocks. The application of *vibration-extinguishers* will not be effective without taking of a frequency spectrum and vibration amplitudes, quantity and duration of shock impulses into consideration. Now the *vibration-extinguishers* are developed, which are various by a assigning and principle of operation. To such

vibration-extinguishers belong and the vibration-extinguishers with stationary magnets considered in the present section.

### 14.4.1. Equation of a vibration-extinguisher motion with stationary magnets

We consider the *pneumatic vibration-extinguisher* (fig. 14.12), consisting from five stationary magnets, between which is situated a gas with some initial pressure [108, 109]. The motion equations of solid bodies of the vibration-extinguisher have the following form:

$$\begin{aligned} M_0 \ddot{x}_0 &= S_0(p_4 - p_1) - kx_0 - h\dot{x}_0 - R_{frict01} \text{sign}(\dot{x}_0 - \dot{x}_1) - R_{frict02} \text{sign}(\dot{x}_0 - \dot{x}_2) - \\ &\quad - R_{frict03} \text{sign}(\dot{x}_0 - \dot{x}_3) - F_{mag01}(\delta_1) + F_{mag03}(\delta_4) ; \\ M_1 \ddot{x}_1 &= S_1(p_1 - p_2) - R_{frict01} \text{sign}(\dot{x}_1 - \dot{x}_0) + F_{mag01}(\delta_1) - F_{mag12}(\delta_2) ; \\ M_2 \ddot{x}_2 &= S_2(p_2 - p_3) - R_{frict02} \text{sign}(\dot{x}_2 - \dot{x}_0) + F_{mag12}(\delta_2) - F_{mag23}(\delta_3) ; \end{aligned} \quad (14.27)$$

$$\begin{aligned} M_3 \ddot{x}_3 &= S_3(p_3 - p_4) - R_{frict03} \text{sign}(\dot{x}_3 - \dot{x}_0) + F_{mag23}(\delta_3) - F_{mag03}(\delta_4) ; \quad \delta_1 = \delta_{10} + x_1 - x_0 ; \\ \delta_2 &= \delta_{20} + x_2 - x_1 ; \quad \delta_3 = \delta_{30} + x_3 - x_2 ; \quad \delta_4 = \delta_{40} + x_0 - x_3 , \end{aligned}$$

where  $M_i$  – masses of stationary magnets;  $S_i$  – cross-sectional area of magnets ( $i=0..3$ );  $k, h$  – coefficients of stiffness and dampings;  $F_{magij}$  – force acting between magnets;  $R_{frict0i}$  – maximal force of static friction;  $\delta_{i0}$  – an initial distance between magnets.

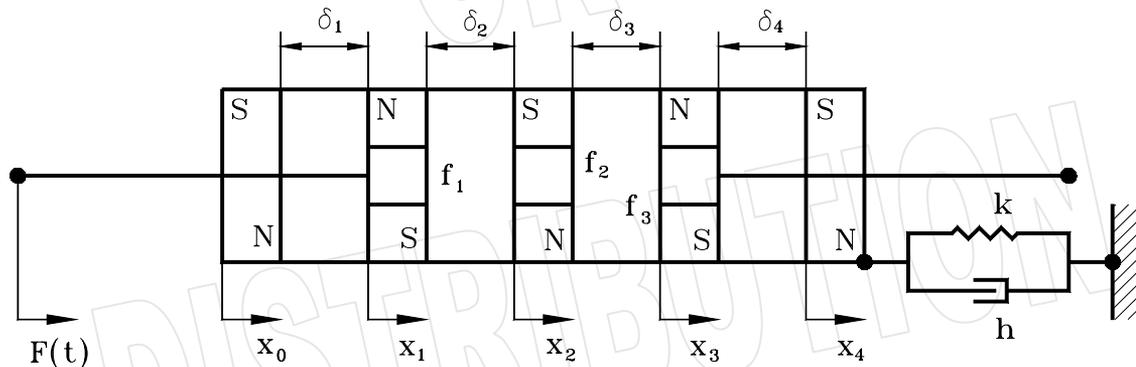


Fig. 14.12. The calculated scheme of pneumatic vibration-extinguisher with stationary magnets

The force acting between two stationary magnets is defined as follows:

$$F_{mag} = cA_c S \sum_{k=1}^{\infty} \frac{1}{2k-1} \left[ \frac{1}{\text{cth}\left(\frac{\pi l(2k-1)}{b}\right) \text{th}\left(\frac{\pi l(2k-1)}{b}\right) + \frac{1}{\mu} \text{sh}\left(\frac{\pi \delta(2k-1)}{2b}\right)} \right]^2, \quad (14.28)$$

where  $c = 5,1 \cdot 10^{-7}$  – constant;  $A_c$  – coercive force of a constant magnet;  $S$  – cross-sectional area;  $l$  – length of a magnet;  $b$  – breadth of a magnet in a longitudinal slit;  $\delta$  – distance between magnets;  $\mu = \frac{\mu_b}{\mu_0}$ ,

$\mu_b$  – magnetic conductivity;  $\mu_0 = 4\pi \cdot 10^{-7}$ . The change of pressure in working chambers of the vibration-extinguisher is defined by the following relations:

$$\dot{p}_1 = \frac{\gamma}{V_1} [RG_{21} \text{sign}(p_2 - p_1) - p_1 S_1 (\dot{x}_1 - \dot{x}_0)];$$

$$\dot{p}_2 = \frac{\gamma}{V_2} [-RG_{21}\text{sign}(p_2 - p_1) + RG_{23}\text{sign}(p_3 - p_2) - p_2 S_2(\dot{x}_2 - \dot{x}_1)]; \quad (14.29)$$

$$\dot{p}_3 = \frac{\gamma}{V_3} [RG_{23}\text{sign}(p_2 - p_3) - RG_{34}\text{sign}(p_3 - p_4) - p_3 S_3(\dot{x}_3 - \dot{x}_2)];$$

$$\dot{p}_4 = \frac{\gamma}{V_4} [RG_{34}\text{sign}(p_3 - p_4) - p_4 S_0(\dot{x}_0 - \dot{x}_3)], \quad (14.30)$$

where  $R$  - gas constant;  $V_i$  - volume of working chambers ( $i=1..4$ );  $V_1 = V_{10} + S_1(x_1 - x_0)$ ;  $V_2 = V_{20} + S_2(x_2 - x_1)$ ;  $V_3 = V_{30} + S_3(x_3 - x_2)$ ;  $V_4 = V_{40} + S_0(x_0 - x_3)$ ;

$$G_{21} = \begin{cases} \mu_1 f_1 T_2 p_2 \varphi(T_2) \Phi\left(\frac{p_1}{p_2}\right); & \text{when } p_1 \geq p_2; \\ \mu_1 f_1 T_1 p_1 \varphi(T_1) \Phi\left(\frac{p_1}{p_2}\right); & \text{when } p_2 > p_1; \end{cases} \quad (14.31)$$

$$G_{23} = \begin{cases} \mu_2 f_2 T_2 p_2 \varphi(T_2) \Phi\left(\frac{p_3}{p_2}\right); & \text{when } p_2 \geq p_3; \\ \mu_2 f_2 T_3 p_3 \varphi(T_3) \Phi\left(\frac{p_2}{p_3}\right); & \text{when } p_3 > p_2; \end{cases} \quad (14.32)$$

$$G_{34} = \begin{cases} \mu_3 f_3 T_3 p_3 \varphi(T_3) \Phi\left(\frac{p_4}{p_3}\right); & \text{when } p_3 \geq p_4; \\ \mu_3 f_3 T_4 p_4 \varphi(T_4) \Phi\left(\frac{p_3}{p_4}\right); & \text{when } p_4 > p_3, \end{cases}$$

where:  $\mu_i$  - coefficient of consumption;  $f_i$  - diameter of a hole in a magnet;  $T_i$ ,  $p_i$  - temperature and pressure in working chambers ( $i=1..4$ );  $\varphi(T) = \sqrt{\frac{2\gamma}{(\gamma-1)RT}}$ ;

$$\Phi\left(\sigma = \frac{p_i}{p_j}\right) = \begin{cases} \sqrt{\sigma^{\frac{2}{\gamma}} - \sigma^{\frac{\gamma+1}{\gamma}}}; & \text{when } 0,528 < \sigma \leq 1; \\ 0,2588; & \text{when } 0 \leq \sigma \leq 0,528. \end{cases}$$

The external perturbation varies under the polyharmonic law, namely:

$$F(t) = \sum_{i=0}^n a_i \sin(b_i t) \quad (14.33)$$

or is a random quantity, distributed under the normal law, namely:

$$F(t) = a_{10} + a_1 \sin(a_3 t) \quad (14.34)$$

The value of a random quantity  $y$ , which answers the normal distribution, is defined as follows:

$$y = \mu + \sigma \left( \sum_{i=1}^{12} R_i - 6 \right) \quad (14.35)$$

where  $\mu$  - average value;  $\sigma^2$  - variance;  $R_i$  - random number of the interval  $[0, 1]$ .

### 14.4.2. Input data for the solution of the problem

For solution of a problem of *definition of vibration-extinguisher oscillations* the **Damper** program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable *F*, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are recorded: number of degrees of freedom, for which into the resultant file the values of unknowns are recorded (*nwr*); a code of a loading (*kodpas*); number of parameters for definition of perturbing force (*np*); a coefficient of stiffness (*astif*) and a coefficient of damping (*adamp*). If *kodpas=1*, then loading is supposed random; otherwise, is supposed the determined.

In the *second* line the next parameters are recorded: a print code of intermediate results (*parameter kprint*); number indicating through what amount of integration steps the results of calculations are recorded (*nstep*); number of integration steps (*ntime*) and an integration step (*dtime*). If *kprint=0*, then the intermediate results are not printed; otherwise they are printed out.

In each subsequent 4 lines, the elements of the arrays **Amase(4)**, **AL(4)**, **Sarea(4)**, **Farea(4)**, **Delta(4)**, **Rforce(4)**, **Ax(4)**, **Av(4)** and **Ap(4)** are coded. Into each line of the datafile the following parameters are recorded: line number; mass of a magnet; length of a magnet; a cross-sectional area of a magnet; the area of a hole in a magnet; initial distance between magnets; the maximal force of static friction; initial *x*-coordinate; initial velocity of a magnet motion; an initial pressure in the working chamber.

Behind of the arrays **Amase**, **AL**, **Sarea**, **Farea**, **Delta**, **Rforce**, **Ax**, **Av** and **Ap** into the datafile the elements of the arrays **Param(np)** are recorded line by line. Into each line of the datafile the line number and elements of the arrays **Param** are recorded. Into each line of the arrays **AL** and **BL** the two geometrical parameters **a<sub>i</sub>** and **b<sub>i</sub>** are recorded accordingly. Into each line of the array **Param** the parameters for definition of perturbing force are recorded. If code of loading *kodpas=0*, then into the array **Param** the parameters (**a<sub>0</sub>**, **b<sub>0</sub>**, **a<sub>1</sub>**, **b<sub>1</sub>**, ...) are recorded, if *kodpas=1*, then into the array **Param** the average values and mean square deviations of a constant constituent of amplitude and frequencies are recorded; in this case is supposed *np=6*.

Behind of the array **Param** into the datafile the elements of the **Mwr(nwr)** array are recorded line by line. Into each line of the datafile the following parameters are recorded: line number and number of a degree of freedom (*number of a body*), whose values of angular displacements, velocities and accelerations will be recorded into the resultant file. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

#### Schematic structure of the datafile:

Text line \*  
*nwr, kodpas, np, astif, adamp, kprint, nstep, ntime, dtime*  
 Text line \*  
 Arrays **Amase(4)**, **AL(4)**, **Sarea(4)**, **Farea(4)**, **Delta(4)**, **Rforce(4)**, **Ax(4)**, **Av(4)**, **Ap(4)**  
 Text line \*  
 Array **Lpar(nbody, 2)**  
 Text line \*  
 Array **Param(np)**  
 Text line \*  
 Array **Mwr(nwr)**

### 14.4.3. Brief description of the Damper program solving the problem

The **Damper** program was programmed on the *Maple*-language; it consists of the basic program and

17 procedures. All procedures can be divided into *three* groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of integration steps of the differential equations. The program calculates *displacements, velocities* and *accelerations* of a *vibration-extinguisher* bodies, and also values of *pressure* in working chambers. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1, file\_rez2, file\_rez3* and *file\_rez4* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of displacements, into the file *file\_rez2* values of velocities, into the file *file\_rez3* values of accelerations of a vibration-extinguisher, at last into the file *file\_rez4* values of pressure in working chambers are recorded accordingly.

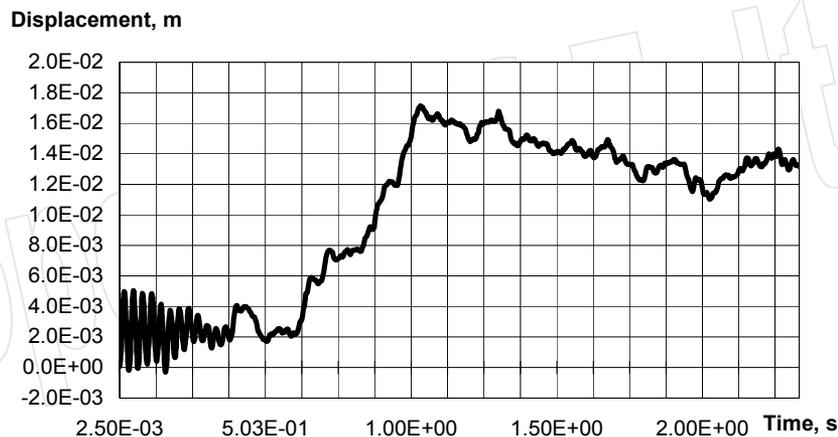
#### 14.4.4. An example of use of the Maple-program Damper

As an example of application of the program *Damper*, the pneumatic vibration-extinguisher with stationary magnets, which is represented on the fig. 14.12, is considered.

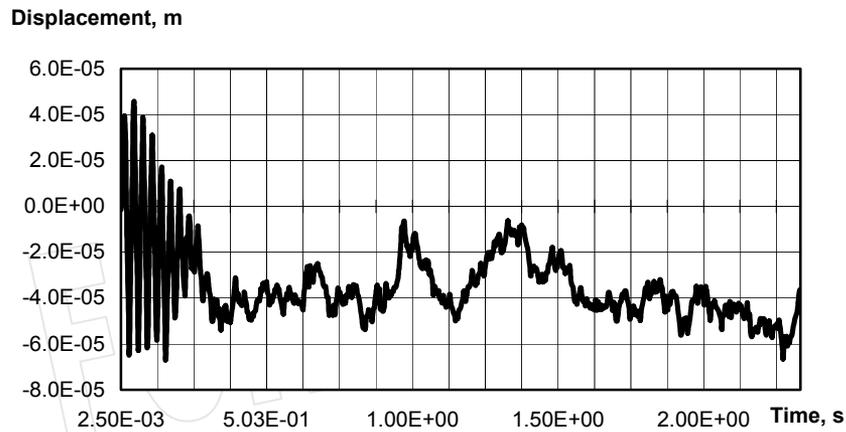
Input data for the test example:  $nwr=4$ ,  $kodpas=1$ ,  $np=6$ ,  $kprint=1$ ,  $nstep=2$ ,  $ntime=5000$ , masses of magnets:  $m_1 = 0,1 \text{ kg}$  ;  $dtime=2 \cdot 10^{(-4)} \text{ s}$ ;  $m_2 = m_3 = m_4 = 0,05 \text{ kg}$  ; coefficient of stiffness  $astif = 10^5 \frac{\text{N}}{\text{m}}$ ; coefficient of damping  $adamp = 10^{-5} \frac{\text{Ns}}{\text{m}}$ ; cross-sectional area of a magnet  $S_i = 1,727 \cdot 10^{-4} \text{ m}^2$  ( $i=0..3$ ); area of a hole in a magnet  $f_i = 2,82 \cdot 10^{-5} \text{ m}^2$ ; initial pressure  $p_{0i} = 5 \cdot 10^5 \text{ Pa}$  ( $i=1..4$ ); initial displacement of a magnet  $x_i(t=0) = 0$ ; initial velocity of a magnet  $\dot{x}_i(t=0) = 0$ . Parameters of the perturbing force:  $a_0$  - average value 0, mean square deviation 0;  $a_1$  - average value 5N, mean square deviation 1.4N;  $a_3$  - average value of frequency of 200 rad/s, mean square deviation 5 rad/s.

#### Results of calculation over the test example:

On the fig. 14.13 the dependences of displacements of the second and the fourth mass of *vibration-extinguisher* in a time are represented; on the fig. 14.14 the dependences of velocities of the second and the fourth mass of vibration-extinguisher in a time are represented, at last on the fig. 14.15 the dependences of pressures in the second and the fourth working chambers of *vibration-extinguisher* in a time are represented.

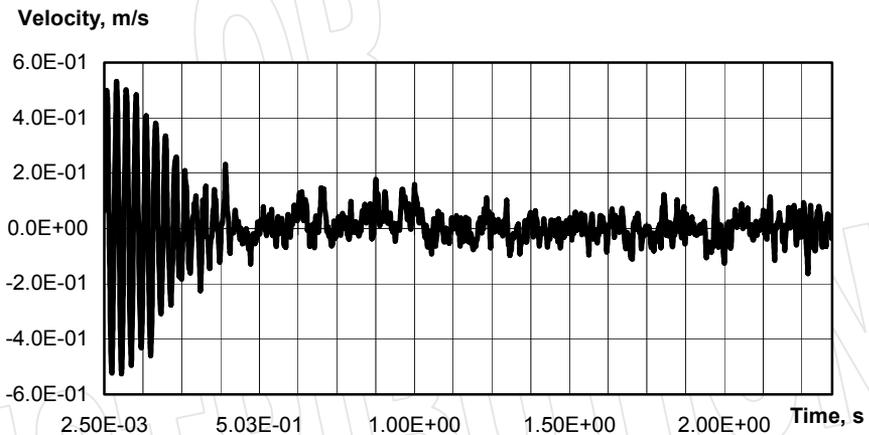


(a)

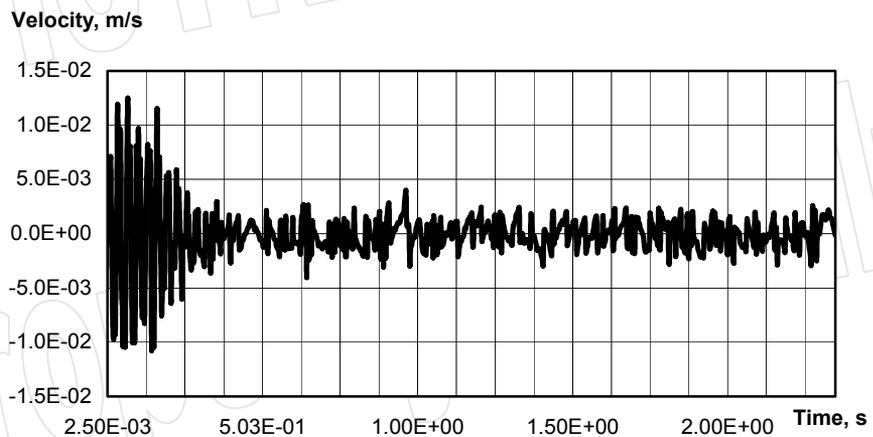


(b)

Fig. 14.13. Dependences of displacements of vibration-extinguisher masses 2 (a) and 4 (b) in a time

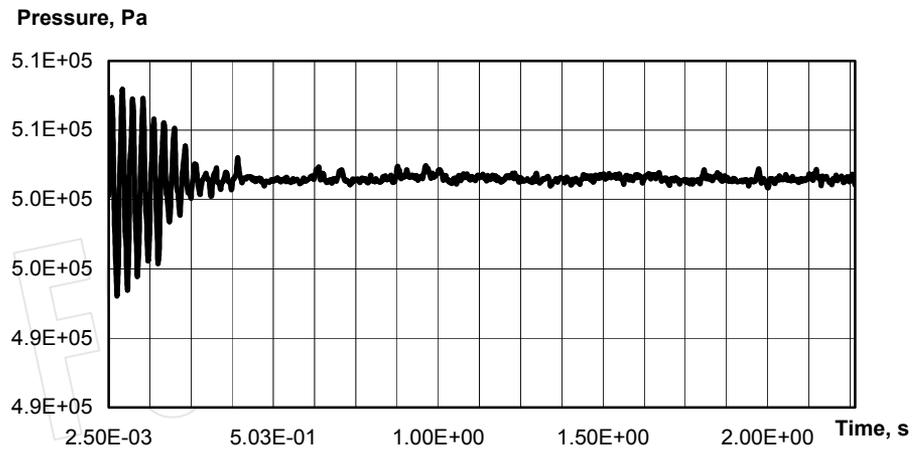


(a)

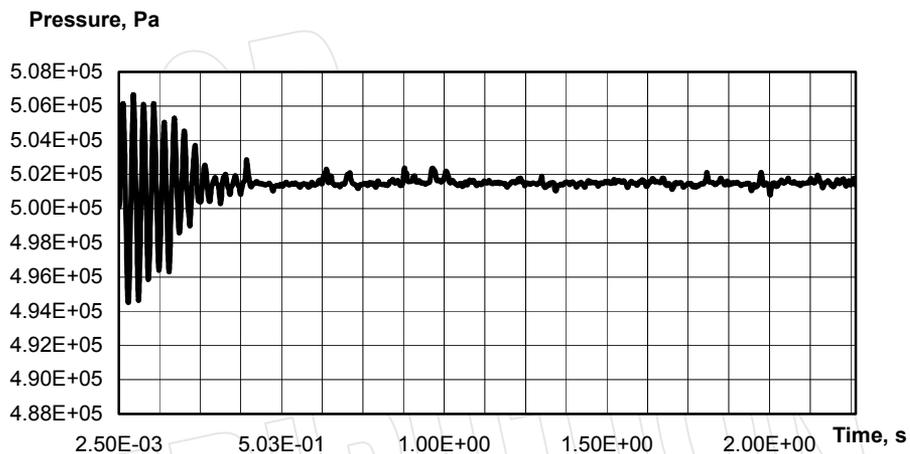


(b)

Fig. 14.14. Dependences of velocities of vibration-extinguisher masses 2 (a) and 4 (b) in a time



(a)



(b)

Fig. 14.15. Dependences of pressure in chambers 2 (a) and 4 (b) of vibration-extinguisher in a time

The *Damper* program is intended for the solution of a problem of *determination of oscillations of pneumatic vibration-extinguisher with stationary magnets*. The source module of the program in Maple-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb11\_4\_new.mws*, *Mb11\_4.dat* and *Mb11\_4\_1.rez* accordingly.

## 14.5. Models of gas motion in a pneumatic system

The modern pneumatic systems consist of the various functional elements such as: compressors, drives, spreaders, gates, receivers, filters, mains etc. The processes happening in these systems, are *quickly mutable*; i.e. *velocity* and *pressure* vary quickly in a time. The sound velocity in a pneumatic system is significant less, in comparison with sound velocity in a hydraulic system, therefore dynamic processes in pneumatic system happen considerably slower. The calculation of physical processes in pneumatic systems depending on properties of the system has the important applied value.

### 14.5.1. Motion of gas in a pneumatic system

The motion of gas in a pneumatic main is supposed by *one-dimensional* and *unsteady*; i.e. all local velocities are considered equal to medial velocity and depend on a time. The pressure also is considered identical in all points of a clear area and depends on longitudinal coordinate of a main and a time. Such gas motion is characterized by a arising of a wave of *heightened* and *lower* pressure, which dilates from a place of change of pressure and strain of walls of the pipeline. The *sound velocity* in a gas (*air*) is defined by the following relation:

$$c = \sqrt{\gamma RT}, \text{ where: } \gamma - \text{isentropic exponent for a gas; } R - \text{gas constant; } T - \text{temperature} \quad (14.36)$$

Then the *motion equation of a gas* in a pneumatic main accepts the following form:

$$\frac{\partial Q}{\partial t} + \frac{\partial p}{\partial x} + \frac{f(Q) c^2}{2\gamma d} \frac{|Q|Q}{p} = 0, \quad (14.37)$$

where  $Q$  - consumption of gas;  $f$  - resistance coefficient,

$$f = \begin{cases} \left[ \frac{0,556}{\lg(\text{Re}/7)} \right]^2; & \text{when } \text{Re} \leq 2300 ; \\ 0,1 \left( \frac{\Delta}{d} + \frac{10\nu}{\text{Re}} \right)^{0,25}; & \text{when } \text{Re} > 2300 , \end{cases} \quad (14.38)$$

$\text{Re}$  - Reynolds number,  $\text{Re} = \frac{ud}{\nu}$ ,  $\nu$  - kinematic viscosity of gas,  $\nu = [1,712 + 5,8 \cdot 10^{-3}(T - 273)] \cdot 10^{-5}$ ;

$\Delta$  - grain of a wall of a main;  $d$  - diameter of a main. For low-pressure pneumatic systems, it is possible to consider gas as *ideal*. Then the *equation of state* accepts the following form:

$$p = \rho RT, \text{ where: } \rho - \text{density of a gas} \quad (14.39)$$

The *Reynolds number* in this case accepts the following form:

$$\text{Re} = \frac{RTdQ}{\nu} \quad (14.40)$$

The *continuity equation* of a gas motion has the following form:

$$\frac{\partial p}{\partial t} + c^2 \frac{\partial Q}{\partial x} = 0 \quad (14.41)$$

As a rule, the differential equations of motion of a fluid in mains are solved by a method of the *characteristics*. The equations system (14.37) and (14.41) is the partial differential equations system of hyperbolic type, i.e. it has two *real characteristics* passing via each point of plane  $X$  and  $t$  (*time*). The *characteristics* - curves, on which exist the *algebraic* or *ordinary differential equations* linking the values of sought functions  $Q$  (*consumption*) and  $p$  (*pressure*). All length of a pneumatic main is divided by elements of length  $\Delta x$ . The unknown variables - a *consumption* and a *pressure* of a gas at a moment  $t+\tau$ , which are defined according to values of these parameters at a moment  $t$  (fig. 14.16).

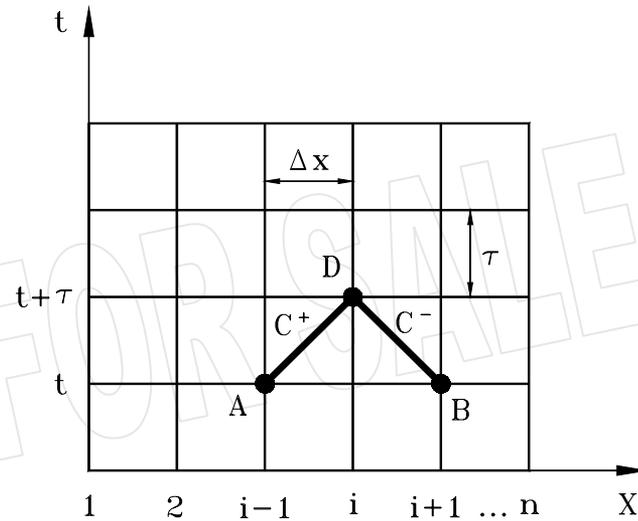


Fig. 14.16. The scheme of definition of parameters of  $D$ -point by method of the characteristics

Pressure and consumption in a point  $D$  at a moment  $t+\tau$  are determined from system of the nonlinear algebraic equations of the following form:

$$C^+ : \Phi_1 = p_D - p_A + c(Q_D - Q_A) + \frac{G}{4} [f(Q_A, p_A) + f(Q_D, p_D)] \frac{|Q_A + Q_D| (Q_A + Q_D)}{p_A + p_D} = 0 ; \quad (14.42)$$

$$C^- : \Phi_2 = p_D - p_B - c(Q_D - Q_B) - \frac{G}{4} [f(Q_B, p_B) + f(Q_D, p_D)] \frac{|Q_B + Q_D| (Q_B + Q_D)}{p_B + p_D} = 0 ; \quad (14.43)$$

$$G = \frac{c^2 \Delta x}{2\gamma d}$$

The system of the nonlinear algebraic equations (14.42) and (14.43) is solved by the Newton method, which in a matrix form accepts the following form:

$$[J]_i \{\Delta Y\}_i = -\{\Phi\}_i, \quad (14.44)$$

where  $\{\Delta Y\}_i^T = [\Delta p_i, \Delta Q_i]$ ;  $\{\Phi\}_i^T = [\Phi_{1i}, \Phi_{2i}]$ ;  $i$  - number of iteration;  $[J]$  - Jacobi matrix,

$$[J] = \begin{bmatrix} \frac{\partial \Phi_1}{\partial p_D} & \frac{\partial \Phi_1}{\partial Q_D} \\ \frac{\partial \Phi_2}{\partial p_D} & \frac{\partial \Phi_2}{\partial Q_D} \end{bmatrix};$$

$$\frac{\partial \Phi_1}{\partial p_D} = 1 + \frac{G}{4(p_A + p_D)^2} \left\{ \frac{\partial f}{\partial p_D} (p_A + p_D) - [f(Q_A, p_A) + f(Q_D, p_D)] \right\} |Q_A + Q_D| (Q_A + Q_D);$$

$$\frac{\partial \Phi_1}{\partial Q_D} = c + \frac{G}{4(p_A + p_D)} \left\{ \frac{\partial f}{\partial Q_D} (Q_A + Q_D) + 2[f(Q_A, p_A) + f(Q_D, p_D)] \right\} |Q_A + Q_D|;$$

$$\frac{\partial \Phi_2}{\partial p_D} = 1 - \frac{G}{4(p_D + p_B)^2} \left\{ \frac{\partial f}{\partial p_D} (p_D + p_B) - [f(Q_B, p_B) + f(Q_D, p_D)] \right\} |Q_D + Q_B| (Q_D + Q_B);$$

$$\frac{\partial \Phi_2}{\partial Q_D} = -c - \frac{G}{4(p_D + p_B)} \left\{ \frac{\partial f}{\partial Q_D} (Q_D + Q_B) + 2[f(Q_D, p_D) + f(Q_B, p_B)] \right\} |Q_D + Q_B|;$$

$$\frac{\partial f}{\partial p_D} = \frac{\partial f}{\partial Re} \frac{\partial Re}{\partial p_D}; \quad \frac{\partial f}{\partial Q_D} = \frac{\partial f}{\partial Re} \frac{\partial Re}{\partial Q_D}; \quad \frac{\partial Re}{\partial p_D} = -\frac{RTd}{\partial p^2} Q_D; \quad \frac{\partial Re}{\partial Q_D} = \frac{RTd}{p_D}; \quad (14.45)$$

$$\frac{\partial f}{\partial p_D} = \begin{cases} \frac{2a}{\ln(10)Re \lg^3(Re/7)} \frac{RTdQ}{vp^2}; & \text{when } Re \leq 2300; \\ \frac{2,5}{Re^2 \left(\frac{\Delta}{d} + 100/Re\right)^{0,75}} \frac{RTdQ}{vp^2}; & \text{when } Re > 2300; \end{cases}$$

$$\frac{\partial f}{\partial Q_D} = \begin{cases} -\frac{2a}{\ln(10)Re \lg^3(Re/7)} \frac{RTd}{\partial p}; & \text{when } Re \leq 2300; \\ -\frac{2,5}{Re^2 \left(\frac{\Delta}{d} + 100/Re\right)^{0,75}} \frac{RTd}{\partial p}; & \text{when } Re > 2300. \end{cases}$$

By solving the equations system (14.45), we define new values of variables:

$$\{Y\}_{i+1} = \{Y\}_i + \{\Delta Y\}_i \quad (14.46)$$

We shall consider now a number of joint nodes of the model of pneumatic system, which represent practically significant interest for a number of the technical appendices.

A joint node of mains (fig. 14.17):

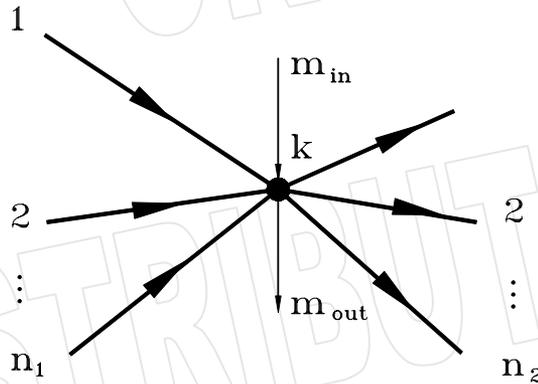


Fig. 14.17. The scheme of joint node of mains with inflow and takeoff of gas

The common node of a pneumatic system is considered, into which come in  $n_1$  and come out  $n_2$  mains. In the common node take place inflow  $m_{in}$  and takeoff  $m_{out}$  of a gas. The equations system of motion of a gas in the common node  $k$  accepts the following form:

$$\Phi_i = p_{i,k} - p_{i,k-1} + c(Q_{i,k} - Q_{i,k-1}) + \frac{G_i}{4} [f(Q_{i,k-1}, p_{i,k-1}) + f(Q_{i,k}, p_{i,k})].$$

$$\frac{|Q_{i,k-1} + Q_{i,k}|(Q_{i,k-1} + Q_{i,k})}{p_{i,k-1} + p_{i,k}} = 0, \quad (i = 1, \dots, n_1); \quad (14.47)$$

$$\Phi_j = p_{j,k} - p_{j,k+1} - c(Q_{j,k} - Q_{j,k+1}) - \frac{G_j}{4} [f(Q_{j,k}, p_{j,k}) + f(Q_{j,k+1}, p_{j,k+1})].$$

$$\frac{|Q_{j,k} + Q_{j,k+1}|(Q_{j,k} + Q_{j,k+1})}{p_{j,k} + p_{j,k+1}} = 0, \quad (j = 1, \dots, n_2). \quad (14.48)$$

For a node of linking of mains the following dependence of material balance is valid:

$$\sum_{i=1}^{n_1} S_i Q_i - \sum_{j=1}^{n_2} S_j Q_j + m_{in} - m_{out} = 0; \quad (14.49)$$

where  $m_i$  – inflow of gas;  $m_{out}$  – takeoff of gas;  $S_i, S_j$  – area of the mains cross-sections. The pressure in the common node is defined by the following relation:

$$p_{i,k} = p_{j,k} = p_k \quad (14.50)$$

The nonlinear system of the algebraic equations is solved by the Newton method:

$$[J]_i \{\Delta Y\}_i = -\{\Phi\}_i, \quad (14.51)$$

where  $\{\Delta Y\}_i^T = [\Delta Q_{1,k}, \Delta Q_{2,k}, \dots, \Delta Q_{n_1,k}, \Delta Q_{1,k}, \Delta Q_{2,k}, \dots, \Delta Q_{n_2,k}, \Delta p_k]$ .

On input into a main a gas consumption is preset, namely:  $q(t, x=0)=Q_1$ . The equations for determination of pressure on input into a main (node 1) have the following form:

$$\Phi = p_1 - p_2 - c(Q_1 - Q_2) - \frac{G}{4} [f(Q_1, p_1) + f(Q_2, p_2)] \frac{|Q_1 + Q_2|(Q_1 + Q_2)}{p_1 + p_2} = 0 \quad (14.52)$$

The equation (14.52) is solved by the Newton method:

$$[J]_i \Delta Q_{1,i} = -\Phi_i, \quad (14.53) - (14.54)$$

where  $[J]_i = \frac{\partial \Phi}{\partial p_1} = 1 - \frac{G}{4(p_1 + p_2)^2} \left( \frac{\partial f}{\partial p_1} (p_1 + p_2) + [f(Q_1, p_1) + f(Q_2, p_2)] \right) |Q_1 + Q_2|(Q_1 + Q_2)$ . The given gas consumption is defined by the following relation:

$$q(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t} \text{ where } a_i - \text{known coefficients } (i=0..3) \quad (14.55)$$

On output from a main the pressure is preset. The equation for definition of a gas consumption on output from a main (node k) has the following form:

$$\Phi = p(t) - p_{k-1} + c(Q_k - Q_{k-1}) + \frac{G}{4} [f(Q_{k-1}, p_{k-1}) + f(Q_k, p_k)] |Q_{k-1} + Q_k| (Q_{k-1} + Q_k) = 0 \quad (14.56)$$

The given equation is solved by the Newton method, namely:

$$[J]_i \Delta Q_{k,i} = -\Phi_i, \quad (14.57) - (14.58)$$

where  $[J]_i = \frac{\partial \Phi}{\partial Q_k} = c + \frac{G}{4} \frac{|Q_{k-1} + Q_k|}{p_{k-1} + p_k} \left( 2[f(Q_{k-1}, p_{k-1}) + f(Q_k, p_k)] + \frac{\partial f}{\partial Q_k} (Q_{k-1} + Q_k) \right)$ . The given pressure  $p(t) = p_k$  depends on a time as the following function:

$$p(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}, \text{ where: } a_i - \text{known coefficients } (i=0..3) \quad (14.59)$$

Nodes of a main are linked with cavities of the cylinder (fig. 14.18):

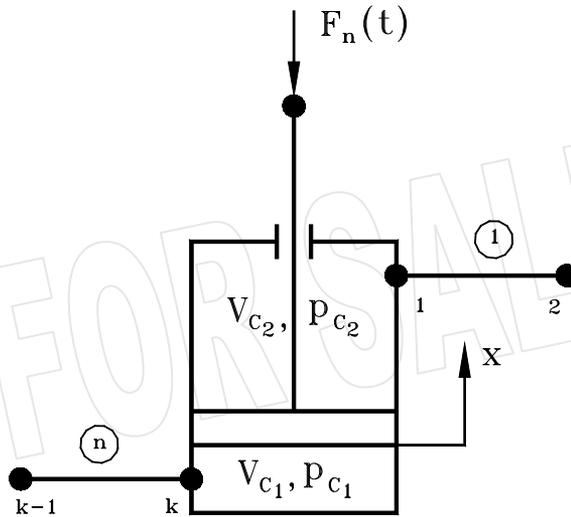


Fig. 14.18. The scheme of a working cavity of the cylinder with a main

According to the first law of thermodynamics, all thermal energy, brought with gas, is spent for change of an internal energy and onto a work of expansion of gas in a cavity of the cylinder. The dependence of pressure in a working cavity of the cylinder is defined by the following relation:

$$\frac{dp_{c_1}}{dt} = \frac{\gamma RT_m G_k}{V_{c_1}} - \frac{\gamma p_{c_1}}{V_{c_1}} \frac{dV_{c_1}}{dt}, \quad (14.60)$$

where  $G_k$  - mass consumption of a gas, being defined under the formula of Saint-Venant and Ventsel. Under condition of  $p_k > p_{c_1}$  takes place the following relation:

$$G_k = \mu_k S_k p_k k_T \phi(\sigma) \quad (14.61) - (14.63)$$

where  $\mu_k$  - coefficient of gas consumption in a node  $k$ ;  $S_k$  - area of an input hole in a node  $k$ ;  $\sigma = \frac{p_{c_1}}{p_k}$ ;

$k_T = \sqrt{\frac{2\gamma}{(\gamma-1) RT_k}}$ ;  $\phi(\sigma) = \sqrt{\sigma^{\frac{2}{\gamma+1}} - \sigma^{\frac{2}{\gamma}}}$ ;  $V_{c_1}$  - volume of gas in a working cavity of the cylinder,

$V_{c_1} = V_{c_{10}} + S_{c_1} x$ ;  $V_{c_{10}}$  - an initial volume of gas in a working cavity of the cylinder;  $S_{c_1}$  - area of the piston;  $x$  - travel of the piston. The dependence of pressure in rod cavity of the cylinder has the following form:

$$\frac{dp_{c_2}}{dt} = -\frac{\gamma RT_{c_2}}{V_{c_2}} G_{c_2} - \frac{\gamma p_{c_2}}{V_{c_2}} \frac{dV_{c_2}}{dt} \quad (14.64) - (14.66)$$

where  $G_{c_2} = \mu_1 S_1 p_{c_2} k_T \phi\left(\sigma = \frac{p_1}{p_{c_2}}\right)$  under condition of  $p_{c_2} > p_1$ ;  $V_{c_2}$  - volume of gas in rod cavity of the cylinder,  $V_{c_2} = V_{c_{20}} - S_{c_2} x$ ;  $V_{c_{20}}$ ,  $S_{c_2}$  - an initial volume of gas and area of the cylinder in rod cavity;

if  $p_1 > p_{c_2}$  then  $G_{c_2} = -\mu_1 S_1 p_1 k_T \phi\left(\sigma = \frac{p_{c_2}}{p_1}\right)$ . Temperature  $T_{c_2}$  can be expressed through a pressure

$p_{c_2}$  on the basis of the equation of an adiabat:

$$\frac{p_{c_2}}{p_1} = \left( \frac{T_{c_2}}{T_1} \right)^{\frac{\gamma}{\gamma-1}} \quad (14.67)$$

The motion equation of the piston has the following form:

$$M_{\text{pist}} \ddot{x} = S_{c_1} p_{c_1} - S_{c_2} p_{c_2} - F_{\text{pist}}(t) - F_{\text{frict}}(\dot{x}) \text{sign} \dot{x} \quad (14.68) - (14.70)$$

where  $F_{\text{pist}}(t)$  - external force acting on the piston,  $F_{\text{pist}}(t) = [b_0 + b_1 \sin(b_2 t)] e^{b_3 t}$ ;  $F_{\text{frict}}(\dot{x})$  - frictional force;  $b_i$  - known coefficients ( $i=0..3$ );  $F_{\text{frict}}(\dot{x}) = F_{\text{frict}0} + h\dot{x}$ ;  $F_{\text{frict}0}$  - maximal force of static friction. The equation for determination of a gas consumption in a node  $k$  of a main has the following form:

$$\Phi_1 = p_{k,t+\tau} - p_{k-1,t} + c(Q_{k,t+\tau} - Q_{k-1,t}) + \frac{G}{4} [f(Q_{k-1,t}, p_{k-1,t}) + f(Q_{k,t+\tau}, p_{k,t+\tau})] \cdot |Q_{k-1,t} + Q_{k,t+\tau}| (Q_{k-1,t} + Q_{k,t+\tau}) = 0 \quad (14.71)$$

The equation for definition of a gas consumption in node 1 of a main has form:

$$\Phi_6 = p_1 - p_2 + c(Q_1 - Q_2) - \frac{G}{4} [f(Q_1, p_1) + f(Q_2, p_2)] |Q_1 + Q_2| (Q_1 + Q_2) = 0 \quad (14.72)$$

On output from a main into the working chamber of the cylinder there are local losses of pressure. Then the dependence between pressure  $p_k$  in the main and pressure  $p_{c_1}$  in the working chamber of the cylinder accepts the next form:

$$\Phi_2 = p_k - p_{c_1} - \frac{1}{2} \frac{RT_k}{p_k} \xi Q^2 \text{sign} Q = 0 \quad \text{where } \xi - \text{coefficient of local losses of pressure} \quad (14.73)$$

On output from the cylinder into the main also there are losses of pressure. The dependence between pressures accepts the following form:

$$\Phi_5 = p_{c_2} - p_1 - \frac{1}{2} \frac{RT}{p_1} \xi Q_1^2 \text{sign} Q_1 = 0 \quad (14.74)$$

The differential equations (14.60), (14.64) and (14.68) are solved by the Euler method, namely:

$$Y_{t+\tau} = Y_t + \tau F_{t+\tau}(Y) \quad (14.75)$$

By applying the Euler method, the equations (14.64) - (14.74) are converted to system of the nonlinear algebraic equations, which are solved by the Newton method, namely:

$$[J]_i \{\Delta Y\} = \{\Phi\}_i, \quad \text{where } \{\Delta Y\}^T = [\Delta p_k \quad \Delta Q_k \quad \Delta p_1 \quad \Delta Q_1 \quad \Delta p_{c_1} \quad \Delta p_{c_2} \quad \Delta x \quad \Delta \dot{x}] \quad (14.76)$$

### 14.5.2. Input data for the solution of the problem

For solution of the equations describing a gas motion in a pneumatic system the *Pneumo* program is used. All data necessary for operation of this program should be recorded into a file in advance; therefore, variable *F*, necessary qualifier of the file, is ascribed. The data in the file are placed in the strict order, namely. In the *first* line the next parameters are recorded: number of mains (*parameter nvam*); number of nodes (*nout*), for which into the file the values of pressure and gas consumption are recorded; number indicating how much of integration steps is demanded for results recording of evaluations into the resultant file (*nstep*).

In the *second* line the next parameters are recorded: a print code of intermediate results (*parameter kprint*); temperature of gas (*temp*); an isentropic exponent (*gama*); inflow and takeoff of gas in the

second node (*amin* and *amout*). If *kprint*=0, then the intermediate results are not printed; otherwise, they are printed out. In the *third* line, the number of iterations for the solution of the nonlinear algebraic equations (*niter*) and a precision of the solution (*toler*) are recorded. In the *fourth* line the number of integration steps in a time (*ntime*) and an integration step (*dtime*) are recorded.

In subsequent *nvam* lines, the elements of the array **Mtop**(*nvam*) are coded. Into each line of the array the number of a main and number of nodes in the given main are recorded. Behind of the array **Mtop** into the datafile the elements of the **Lout**(*nout*, 2) array are recorded line by line. Into each line of the datafile the line number and elements of the **Lout** array are recorded. Into the **Lout** array the are recorded the numbers of mains and numbers of nodes for which the values of pressure and a gas consumption are recorded into the resultant file.

Behind of the array **Lout** into the datafile the elements of the four arrays **Plotas**(*nvam*), **Sd**(*nvam*), **P0**(*nvam*) and **V0**(*nvam*) are recorded line by line. Into each line of the datafile the following parameters are recorded: the number of a main; a cross-sectional area of a main; a grain of a wall of a main; an initial pressure in a main and also an initial gas consumption.

Behind of the arrays **Plotas**(*nvam*), **Sd**(*nvam*), **P0**(*nvam*) and **V0**(*nvam*) into the datafile the elements of the arrays **Param**(5, 4) are recorded line by line. Into each line of the datafile the line number of this array and four elements of the **Param** array are recorded. The structure of the array

**Param**(5, 4) has the following view:  $\text{Param}(5,4) = \begin{bmatrix} b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ b_0 & b_1 & b_2 & b_3 \\ a_0 & a_1 & a_2 & a_3 \\ a_0 & a_1 & a_2 & a_3 \end{bmatrix}$ . Into the *first* and the *second*

lines of the array **Param** the coefficients **b<sub>i</sub>** for definition of external forces acting onto the first and the second cylinder, are recorded. Into the *third*, the *fourth* and the *fifth* lines of the array **Param** are recorded the coefficients **a<sub>i</sub>** for definition of gas consumption and pressures in receiver, and also at the end of the third and the fifth mains.

Behind of the array **Param** into the datafile the elements of the **Param1**(2,12) array are recorded line by line. Into each line, the line number and the parameters of cylinders are recorded such as: reduced mass of propellant masses to a rod of the cylinder; the area of the piston and an area of the piston in rod cavity; initial volume of gas of a working cavity and rod cavity of the cylinder; the maximal piston stroke; the maximal force of static friction; coefficient of damping; initial displacement; velocity of the piston; initial pressures in the working chambers and rod chambers of the cylinder. In the datafile between the recorded arrays the lines of the comments with names of appropriate arrays are located. At reading of the information from the datafile these text lines are skipped.

**Schematic structure of the datafile:**

Text line \*  
*nvam, nout, nstep, kprint, temp, gama, amin, amout, niter, toler, ntime, dtime*  
 Text line \*  
 Array **Mtop**(*nvam*)  
 Text line \*  
 Array **Lout**(*nout*, 2)  
 Text line \*  
 Arrays **Plotas**(*nvam*), **Sd**(*nvam*), **P0**(*nvam*), **V0**(*nvam*)

Text line \*  
 Array *Param*(5, 4)  
 Text line \*  
 Array *Param1*(2, 12)

### 14.5.3. Brief description of the Pneumo program solving the problem

The *Pneumo* program was programmed on the *Maple*-language; it consists of the basic program and 19 procedures. All procedures can be divided into three groups: procedures for data entry, for calculations and for output of results. Memory size necessary for the solution of a concrete problem, and a time of its solution depend on the used number of elements and time of integration. The program calculates values of pressures and gas consumption in each main of a pneumatic system; pressures in the working chambers and rod chambers of cylinders, and also displacements and velocities of pistons. The calculation results are output on the monitor and are recorded into files, for that to variables *file\_rez1*, *file\_rez2*, *file\_rez3* and *file\_rez4* of the program must be ascribed the qualifiers of target files. Into the file *file\_rez1* values of pressure in the given nodes of mains, into the file *file\_rez2* values of gas consumption in the given nodes of mains, into the datafile *file\_rez3* parameters of the first cylinder, at last into the file *file\_rez4* parameters of the second cylinder are recorded accordingly.

### 14.5.4. An example of use of the Maple-program Pneumo

A gas motion in the pneumatic system represented on the fig. 14.19 is considered.

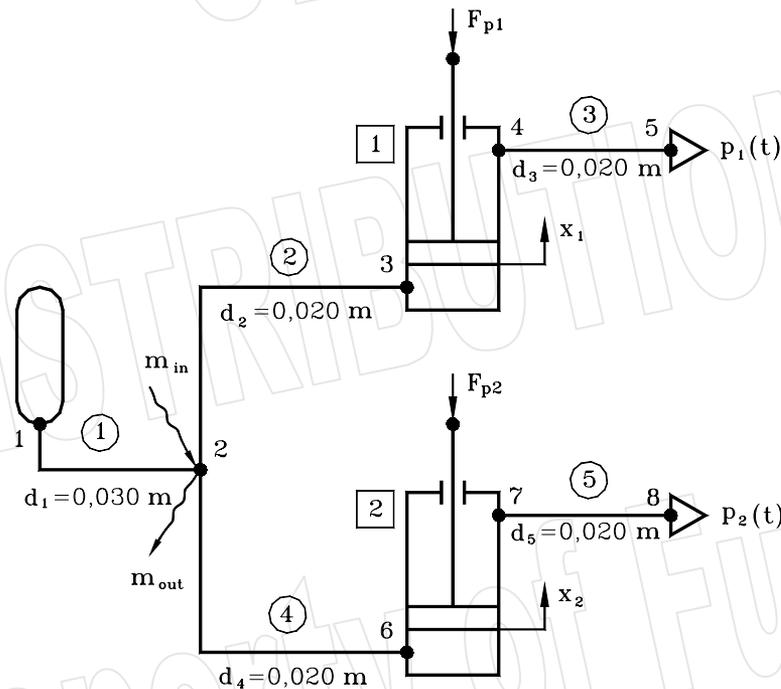


Fig. 14.19. The scheme of the explored pneumatic system

The explored pneumatic system has the following characteristics. In node 1 the gas consumption is preset:  $q(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}$ ,  $a_0 = 10 \frac{\text{kg}}{\text{s} \cdot \text{m}^2}$ ,  $a_1 = a_2 = a_3 = 0$ . In node 2 inflow and takeoff of gas  $a_{in} = 5 \text{ kg/s}$  and  $a_{out} = 0 \text{ s}$  are preset accordingly. In nodes 5 and 8 the pressure is preset:  $p(t) = [a_0 + a_1 \sin(a_2 t)] e^{a_3 t}$ ,  $a_0 = 1,0 \cdot 10^5 \text{ MPa}$ ,  $a_1 = a_2 = a_3 = 0$ .

*Input data for the test example:*  $nvam=5$ ,  $nout=30$ ,  $nstep=1$ ,  $kprint=1$ ,  $gama=1.4$ ,  $temp=293$  K,  $niter = 100$ ,  $toler=10^{(-6)}$ ,  $ntime=300$ ,  $dtime=10^{(-4)}$  s; grain of mains:  $\Delta_i = 1 \cdot 10^{-4}$  m ( $i=1..5$ ); to a rod of the *first* cylinder the force affects:  $F_{pist1}(t) = [b_0 + b_1 \sin(b_2 t)]e^{b_3 t}$ ,  $b_0 = 100$  N ,  $b_1 = b_2 = b_3 = 0$ ; to a rod of the *second* cylinder a force affects:  $F_{pist2}(t) = [b_0 + b_1 \sin(b_2 t)]e^{b_3 t}$ ,  $b_0 = 100$  N ,  $b_1 = b_2 = b_3 = 0$ .

Results of calculation over the test example:

On the *figs. 14.20 and 14.21* the graphs of change of pressure and gas consumption in nodes of the mains 1-2-3 in a time dependence are represented.

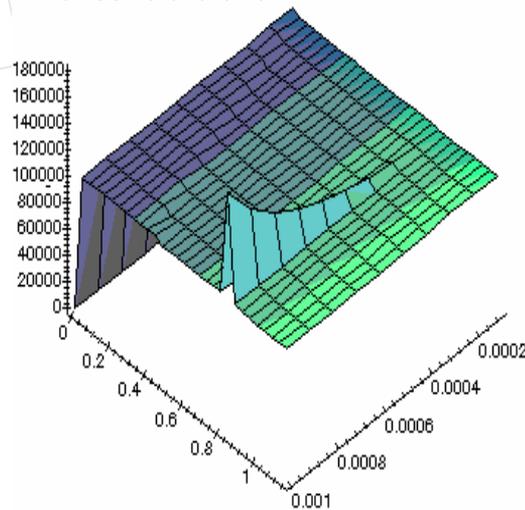


Fig. 14.20. Dependence of gas pressure in a time in nodes of mains 1-2-3

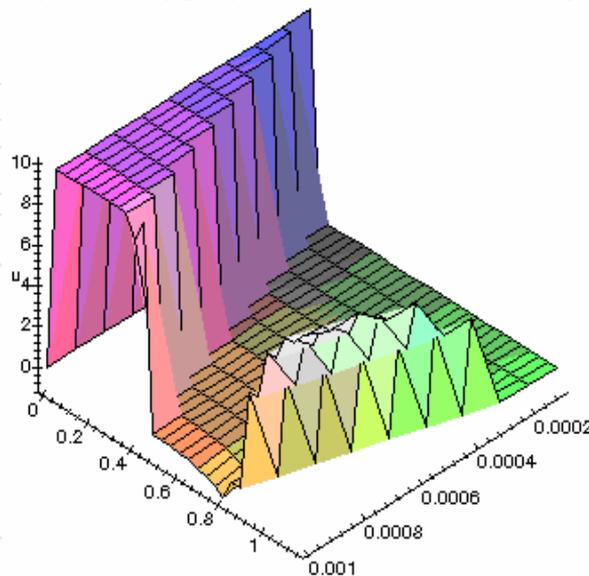


Fig. 14.21. Dependence of gas consumption in a time in nodes of mains 1-2-3

The *Pneumo* program is intended for the solution of motion equations of a gas in pneumatic systems corresponding to the considered model. The source module of the program in *Maple*-language, initial data for the test example, and also outcomes of its solution are represented in the PROBLEMS directory of archive attached to the book in files *Mb11\_5\_new.mws*, *Mb11\_5.dat* and *Mb11\_5\_1.rez* accordingly.

# Chapter 15.

## Application of Maple 6 for solution of optimization problems

The problem of minimizing the given function

$$\min f(\{\mathbf{x}\}) \tag{15.1}$$

subject to given constraints

$$\mathbf{h}_i(\{\mathbf{x}\}) = 0, i = 1, \dots, m_h \tag{15.2}$$

$$\mathbf{g}_j(\{\mathbf{x}\}) \leq 0, j = 1, \dots, m_g \tag{15.3}$$

is called the *general constrained optimization problem*. Where  $\{\mathbf{x}\}$  denotes the vector of variables with components  $\mathbf{x}_j, j=1..n$ . Function  $f(\{\mathbf{x}\})$  appearing in (15.1) is called the *objective function*. Functions  $\mathbf{h}_i(\{\mathbf{x}\})$  ( $i = 1, \dots, m_h$ ) and  $\mathbf{g}_j(\{\mathbf{x}\})$  ( $j = 1, \dots, m_g$ ) are called *equality constraints* and *inequality constraints*. An optimization problem is said to be *linear* when both the *objective function* and the *constraints* are *linear functions* of variables  $\mathbf{x}_i$ , i.e., they can be expressed in the following form:

$$f(\{\mathbf{x}\}) = \{\mathbf{c}\}^T \{\mathbf{x}\} = \sum_{i=1}^n c_i x_i \tag{15.4}$$

*linear optimization problems* are solved by the branch of mathematical programming called *linear programming*. The optimization problem is said to be *nonlinear* if either the *objective function* or the *constraints* are *nonlinear functions* of variables.

### 15.1. Search methods for unconstrained optimization

In this chapter, the problem of minimizing function  $f(\{\mathbf{x}\})$  of real variables  $\mathbf{x}_i$  under the assumption that no constraint is imposed on the values of  $\mathbf{x}_i$  is considered. A direct search method is a method which relies only on evaluating  $f(\{\mathbf{x}\})$  at the sequence of points  $\{\mathbf{x}_1\}, \{\mathbf{x}_2\}, \dots$  and comparing values, in order to reach the optimal point  $\{\mathbf{x}^*\}$ . Direct search methods are commonly used in the following circumstances: function  $f(\{\mathbf{x}\})$  is not differentiable or is subject to random error; derivatives  $\frac{\partial f}{\partial \mathbf{x}}$  are discontinuous or their evaluation is much more difficult than the evaluation of the function  $f(\{\mathbf{x}\})$  itself. The fundamental problem in the design of a direct method is to determine the point  $\{\mathbf{x}_{k+1}\}$  given points  $\{\mathbf{x}_1\}, \dots, \{\mathbf{x}_k\}$  and function values  $f(\{\mathbf{x}_1\}), \dots, f(\{\mathbf{x}_k\})$ .

### 15.1.1. Nelder and Mead's method

The *Nelder and Mead's* method is one of the most efficient pattern search methods currently available and has been found to work particularly well if the number of variables does not exceed five or six. Consider the problem of minimizing  $f(\mathbf{x})$ . Let  $\{\mathbf{x}_1\}$  be an initial estimate of  $\{\mathbf{x}^*\}$  and let the vertices of the initial simplex be  $\{\mathbf{x}_1\}, \dots, \{\mathbf{x}_{n+1}\}$  where  $\{\mathbf{x}_{j+1}\} = \{\mathbf{x}_1\} + h_j \{\mathbf{e}_j\}$  ( $j = 1, \dots, n$ ) the  $\{\mathbf{e}_j\}$  are usual unit coordinate vectors and scalars  $h_j$  are chosen to equalize, as far as possible, the quantities.

$$|f(\{\mathbf{x}_1\} + h_j \mathbf{e}_j) - f(\{\mathbf{x}_1\})| \tag{15.5}$$

In the current simplex, let  $\{\mathbf{x}_h\}$  be the vertex with the highest function value;  $\{\mathbf{x}_s\}$  be the vertex with the second highest function value;  $\{\mathbf{x}_l\}$  be the vertex with the lowest function value;  $\{\mathbf{x}_c\}$  be the centroid of all the vertices except  $\{\mathbf{x}_h\}$ , i.e.:

$$\{\mathbf{x}_c\} = \frac{1}{n} \sum_{j=1}^{n+1} \{\mathbf{x}_j\}; \quad j \neq h \tag{15.6}$$

Let  $y = f(\{\mathbf{x}\})$ ,  $y_h = f(\{\mathbf{x}_h\})$ ,  $y_s = f(\{\mathbf{x}_s\})$ ,  $y_l = f(\{\mathbf{x}_l\})$ ,  $y_c = f(\{\mathbf{x}_c\})$ . Then the procedure recommended by Nelder and Mead method for minimization of  $f(\{\mathbf{x}\})$  is as follows:

1. Choose the vertices of the initial simplex as described above and evaluate  $f(\{\mathbf{x}\})$  at each vertex.
2. *Reflection.* Reflect  $\{\mathbf{x}_h\}$  (fig. 15.1) using a reflection factor  $\alpha > 0$ , i.e. find  $\{\mathbf{x}_0\}$  so that

$$\{\mathbf{x}_r\} = (1 + \alpha)\{\mathbf{x}_c\} - \alpha\{\mathbf{x}_h\} \tag{15.7}$$

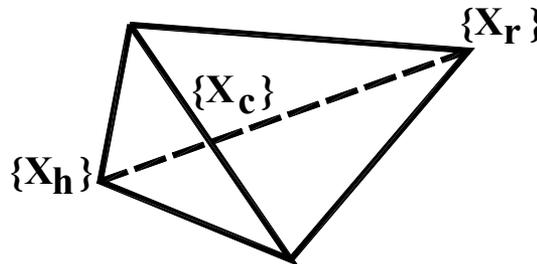


Fig. 15.1. Reflection of vertex  $\{\mathbf{x}_h\}$  in non-regular simplex

1. 1. If  $y_l \leq y_r \leq y_s$ , replace  $\{\mathbf{x}_h\}$  by  $\{\mathbf{x}_r\}$  and return to step 2.
2. 2. *Expansion.* If  $y_r < y_l$ , expand the simplex (fig. 15.2) using expansion factor  $\gamma > 1$ , i.e. find  $\{\mathbf{x}_e\}$ , i.e. find  $\{\mathbf{x}_e\}$  so that

$$\{\mathbf{x}_e\} = \gamma\{\mathbf{x}_r\} + (1 - \gamma)\{\mathbf{x}_c\} \tag{15.8}$$

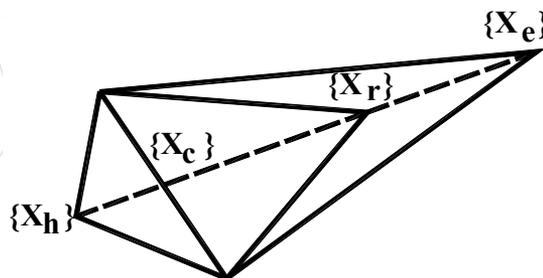


Fig. 15.2. Expansion of non-regular simplex

- a) If  $y_e < y_l$ , replace  $\{x_h\}$  by  $\{x_e\}$  and return to step 2.
  - b) If  $y_e < y_l$ , replace  $\{x_h\}$  by  $\{x_0\}$  and return step 2.
1. **Contraction.** If  $y_r < y_s$ , contract the simplex, using a contraction factor  $\beta(0 < \beta < 1)$ . There are two cases to consider:
- a) If  $y_r < y_h$  (fig.15.3), and find  $\{x_e\}$  so that

$$\{x_e\} = \beta\{x_r\} + (1-\beta)\{x_c\} \tag{15.9}$$

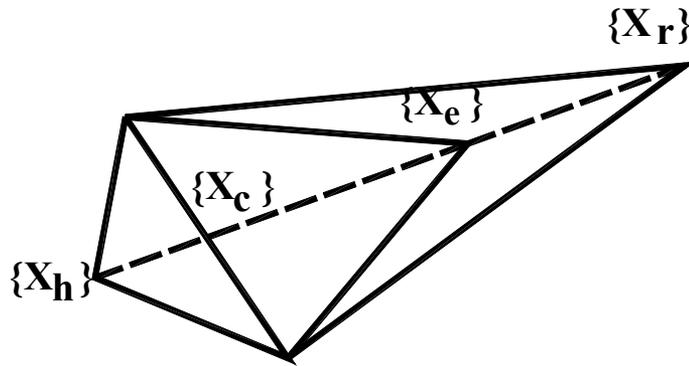


Fig. 15.3. Contraction of non-regular simplex when  $y_s < y_r < y_h$

- b) If  $y_r < y_h$  (fig. 15.4), find  $\{x_e\}$  so that

$$\{x_e\} = \beta\{x_h\} + (1-\beta)\{x_c\} \tag{15.10}$$

Whether 5a or 5b is used, there are again two cases to consider:

- a) If  $y_e < y_h$  and  $y_e < y_r$ , replace  $\{x_h\}$  by  $\{x_e\}$  and return to step 2.
- b) If  $y_e \geq y_h$  or  $y_e < y_r$  reduce the size of the simplex by halving the distances from  $\{x_l\}$  and return to step 2.

Nelder and Mead method suggests values of  $\alpha=1$ ,  $\beta=0.5$  and  $\gamma=2$  for the reflection contraction and expansion factors, respectively.

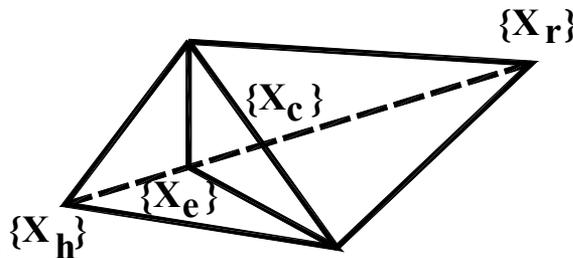


Fig. 15.4. Contraction of non-regular simplex when  $y_r \geq y_h$

A convenient convergence criterion is that the calculation terminate when the standard deviation of  $y_1, y_2, \dots, y_{n+1}$  is less than some prescribed value  $\epsilon > 0$ , i.e. when

$$s = \sum_{i=1}^{n+1} \sqrt{\frac{1}{n}(y_i - \bar{y})^2} < \epsilon, \text{ where } \bar{y} = \frac{1}{n+1} \sum_{i=1}^{n+1} y_i \tag{15.11}$$

### 15.1.1.1. Input data for the solution of the problem

To solve the unconstrained *optimization problem* by Nelder and Mead's method, the *Nelder\_Mead* program is used. All data necessary for operation of this program should be recorded in a file, in advance, therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[i])$  must be written in the procedure *F\_Funk*. In the first line, the number of variables (parameter *nvar*) is coded. In the second line, a print code of intermediate results (*kprint*) parameters *alfa*, *beta*, *gamma* and a precision of the solution (*toler*) are coded. In each subsequent war lines, the number of the variable and the initial values of variable **X0** are coded. When reading the information from the data file, text lines are skipped.

The schematic structure of the datafile:

Text line \*  
*nvar*  
*kprint, astep, alfa, beta, gama, toler*  
 Text line \*  
 Vector **X0**(*nvar*)

### 15.1.1.2. Brief description of the Nelder\_Mead program solving the problem

The *Nelder\_Mead* program was programmed on the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depend on the used number of variables and the preciseness of the solution. Calculation results are output on the monitor and are recorded into a file, therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.1.1.3. An example of use of the Maple-program Nelder\_Mead

The objective function is

$$\min f(\{x\}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (\mathbf{u}[1] + 10 * \mathbf{u}[2])^2 + 5(\mathbf{u}[3] - \mathbf{u}[4])^2 + (\mathbf{u}[2] - 2 * \mathbf{u}[3])^4 + 10(\mathbf{u}[1] - \mathbf{u}[4])^4;$$

Number of variables is **nvar = 4**; **astep = 1.0**; **alfa = 1.0**; **beta = 0.5**; **gama = 2.0**; **toler = 10<sup>-9</sup>**. Initial solution **x0[1]=3.0**; **x0[2]=-1.0**; **x0[3]=0.0**; **x0[4]=1.0**. Result: **x[1]=-7.252045\*10<sup>^-5</sup>**; **x[2] = -7.455866\*10<sup>^6</sup>**; **x[3] = 1.549169\*10<sup>^-5</sup>**; **x[4] = 3.464412\*10<sup>^-5</sup>**; **min f(x1, x2, x3, x4) = 1.493281\*10<sup>^-9</sup>**. The number of iteration is equal to 77, the number of calculations of objective function equals 200. The *Nelder\_Mead* program solves nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is represented in datafile **Mb12\_1\_new.mws** of the **PROG\_EXE** directory, while initial data for the test example, and also outcomes of its solution are represented in datafiles **Mb12\_2.dat** and **Mb12\_2.rez** accordingly of the **PROBLEMS** directory of archive attached to the present book.

### 15.1.2.Hooke and Jeeves' method

Hooke and Jeeves' method is one of the most widely used direct search methods. It attempts in a simple though ingenious way to find the most profitable search directions. We shall consider the problem of minimizing  $f(\{x\})$ . We choose initial base point  $\{b_1\}$  and step lengths  $h_i$  for the

respective variable  $x_i$ . For greater numerical accuracy, it is advisable to choose the  $h_i$  so as to equalize, as far as possible, the quantities

$$|f(\{b_1\} + h_i\{e_i\}) - f(\{b_1\})|$$

these are the magnitudes of the changes in  $f(\{b_1\})$  due to the change of one step length in each variable in turn. After  $f(\{b_1\})$  has been evaluated, the method proceeds by a sequence of exploratory and pattern moves. If an exploratory move leads to a decrease in the value of  $f(\{x\})$ , it is called a success; otherwise, it is called a failure. A pattern move is not tested for success or failure.

**Exploratory moves.** The purpose of an exploratory move is to acquire information about  $f(x)$  in the neighborhood of the current base point. The procedure for an exploratory move about point  $\{b_1\}$  is as follows:

- a) evaluate  $f(\{b_1\} + h_1\{e_1\})$ . If the move from  $\{b_1\}$  to  $\{b_1\} + h_1\{e_1\}$  is a success, replace the base point  $\{b_1\}$  by  $\{b_1\} + h_1\{e_1\}$ . If it is a failure, evaluate  $f(\{b_1\} - h_1\{e_1\})$ . If this move is a success, replace  $\{b_1\}$  by  $\{b_1\} - h_1\{e_1\}$ . If it is another failure, retain the original base point  $\{b_1\}$ .
- b) repeat a) for the variable  $x_2$  by considering variations  $\pm h_2\{e_2\}$  from the point which results from a). Apply this procedure to each variable in turn, finally arriving at a new base point  $\{b_2\}$  after  $(2n + 1)$  function evaluations at most, including  $f(\{b_1\})$ .
- c) if  $\{b_1\} = \{b_2\}$ , halve each of the step lengths  $h_1$  and return to a). The calculations terminate when the step lengths have been reduced to some prescribed level. If  $\{b_2\} \neq \{b_1\}$ , make a pattern move from  $\{b_2\}$ .

**Pattern moves and subsequent moves.** A pattern move attempts to speed up the search by using information acquired about  $f(\{x\})$ . We shall denote by  $\{p_1\}$ ,  $\{p_2\}$ , ... the points reached by successive pattern moves. It seems sensible to move from  $\{b_2\}$  in the direction of  $(\{b_2\} - \{b_1\})$  since a move in this direction has already led to a decrease in value  $f(\{x\})$ . The procedure for a pattern move from  $\{b_2\}$  is therefore as follows:

- a) move from  $\{b_2\}$  to  $\{p_1\} = 2\{b_2\} - \{b_1\}$  and continue with a new sequence of exploratory moves about  $\{p_1\}$ .
- b) if the lowest function value obtained during the pattern and exploratory moves of  $\{p_i\}$  is less than  $f(\{b_2\})$ , then a new base point  $\{b_3\}$  has been reached. In this case, return to  $\{p_1\}$  with all suffices increased by unity. Otherwise, abandon the pattern move from  $\{b_2\}$  and continue with a new sequence of exploratory moves about  $\{b_2\}$ .

### 15.1.2.1. Input data for the solution of the problem

To solve the unconstrained optimization problem by Hooke and Jeeves' method the *Hooke\_Jeeves* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order, namely. The expression of the objective function  $f(u[i])$  must be written in the procedure  $F\_FUNK$ . In the first line, the number of variables (parameter  $nvar$ ) is coded. In the second line, a print code of intermediate results ( $kprint$ ), step ( $astep$ ) and a precision of the solution ( $toler$ ) are coded. When reading the information from the data file text lines are skipped. In each subsequent war lines, the number of the variable and the initial values of variable  $X0$  are coded.

The schematic structure of the datafile:

Text line \*  
*nvar*  
*kprint, astep, alfa, beta, gama, toler*  
 Text line \*  
 Vector  $X0(nvar)$

**15.1.2.2. Brief description of the Hooke\_Jeeves program solving the optimization problem**

The *Hooke\_Jeeves* program was programmed in the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

**15.1.2.3. An example of use of the Maple-program Hooke\_Jeeves**

Objective function is

$$\min f(\{x\}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (u[1] + 10 * u[2])^2 + 5(u[3] - u[4])^2 + (u[2] - 2 * u[3])^4 + 10(u[1] - u[4])^4;$$

The number of variables is  $nvar = 4$ ;  $astep = 1.0$ ;  $toler = 10^{-9}$ . Initial solution  $x0[1]=3$ ;  $x0[2]=-1$ ;  $x0[3]=0$ ;  $x0[4]=1$ . The result is the following:  $x[1]=0$ ,  $x[2]=0$ ,  $x[3]=0$ ,  $x[4]=0$ ,  $\min f(x_1, x_2, x_3, x_4)=0$ . The number of iteration equals 3, the number of calculation of objective function equals 182. The *Hooke\_Jeeves* program solves nonlinear unconstrained optimization problems.

The source module of the program in *Maple*-language is represented in file *Mb12\_2\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_2.dat* and *Mb12\_2.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

**15.1.3. Davidon's cubic interpolation method**

Consider the *linear search problem* of minimizing  $f(\{x\})$  along the line  $\{x\} = \{x_k\} + \alpha\{s\}$ , where  $\{x_k\}$  is the current point and  $\{s\}$  is a given search direction. Suppose that  $f_0 = f(\{x_k\})$  and  $f_\alpha = f(\{x_k\} + \alpha\{s\})$  are known, where  $\alpha$  is a given value. Suppose that the *directional derivatives*

$$G_0 = \left\{ \frac{\partial f(\{x_k\})}{\partial x} \right\}^T \{S\} = \{g(\{x_k\})\}^T \{S\}; \tag{15.12}$$

$$G_\alpha = \left\{ \frac{\partial f(\{x_k\} + \alpha\{S\})}{\partial x} \right\}^T \{S\}, \text{ where } G_0 < 0 \tag{15.13}$$

Calculations for this method are in three distinct stages:

- a) The order of magnitude of  $\alpha_m$  the minimizing value of  $\alpha$  is estimated.
- b) Upper and lower bounds are found for  $\alpha_m$ .

c) Cubic interpolation is used to find more precise bounds on  $\alpha_m$ .

With regard to a), it is usual to take the parameter  $\beta$  as an initial approximation to  $\alpha_m$ , where

$$\beta = \min \left[ \chi, -\frac{2(f_0 - f_e)}{G_0} \right] \quad (15.14)$$

where  $\chi$  is some magnitude for the problem (usually,  $\chi = 2$  and  $f_e$  is a preliminary estimate, low rather than high, of  $f(\{x_k\} + \alpha_m\{S\})$ ). Expression  $[-2(f_0 - f_e)/G_0]$  in equation (15.14) is equal to  $\alpha_m$  in the case where  $f(\{x\})$  is a quadratic function of  $\{x\}$  and where  $f_m$  is an exact estimate of the minimum value of  $f(\{x\})$  on the line  $\{x\} = \{x_k\} + \alpha\{S\}$ . To implement stages (b) and (c) Davidon approximates the function  $f(\{x_k\} + \alpha\{S\})$  by cubic  $\varphi(\alpha)$  which satisfies next conditions:

$$\varphi(0) = f_0, \varphi(\beta) = f_\beta, \frac{d\varphi(0)}{d\alpha} = G_0; \frac{d\varphi(\beta)}{d\alpha} = G_\beta \quad (15.15)$$

Since  $G_0 < 0$ , there is certainly a minimum of  $\varphi(\alpha)$  between  $\alpha = 0$  and  $\alpha = \beta$  if  $G_\beta > 0$  or if  $f_\beta > f_0$ . When neither of these conditions is valid, point  $\{x_k\} + \beta\{S\}$  is replaced by point  $\{x_k\} + 2\beta\{S\}$ . Repeating this procedure the minimum of  $\varphi(\alpha)$  is obtained, and the cubic interpolation starts. The interpolation formula for stage (c) is obtained as follows. Assume that the approximating cubic is given by the next formulae:

$$\varphi(\alpha) = \varphi_0 + \varphi_1\alpha + \varphi_2\alpha^2 + \varphi_3\alpha^3 \quad (15.16)$$

which already satisfies the *first* and *third* of conditions (15.15), i.e.  $\varphi_0 = f_0$ ,  $\varphi_1 = G_0$ . Then

$$\frac{d\varphi}{d\alpha} = \varphi_1 + 2\varphi_2\alpha + 3\varphi_3\alpha^2 \quad (15.17)$$

using the *second* and *fourth* of conditions (15.15), when  $\alpha = \beta$ , we obtain two equations for  $\varphi_2$  and  $\varphi_3$ :

$$\left\{ \begin{array}{l} \beta^2\varphi_2 + \beta^3\varphi_3 = f_\beta - f_0 - G_0\beta \\ 2\beta\varphi_2 + 3\beta^3\varphi_3 = G_\beta - G_0 \end{array} \right\}. \quad (15.18)$$

The solution of system algebraic equation (15.18) is equal

$$\varphi_2 = -(G_0 + Z)/\beta; \quad (15.19)$$

$$\varphi_3 = \frac{(G_0 + G_\beta + 2Z)}{3\beta^2}, \text{ where } Z = \frac{3}{\beta}(f_0 + f_\beta) + G_0 + G_\beta. \quad (15.20 - 15.21)$$

The identity (15.17) now becomes as follows:

$$\frac{d\varphi}{d\alpha} = G_0 - 2(G_0 + Z)\frac{\alpha}{\beta} + (G_0 + G_\beta + 2Z)\frac{\alpha^2}{\beta^2} \quad (15.22)$$

When find  $\alpha_m$ , we set  $\frac{d\varphi}{d\alpha} = 0$  and solve for  $\alpha/\beta$ :

$$\frac{\alpha_m}{\beta} = \frac{\mathbf{G}_0 + \mathbf{Z} \pm \omega}{\mathbf{G}_0 + \mathbf{G}_\beta + 2\mathbf{Z}}, \text{ where } \omega = \sqrt{\mathbf{Z}^2 + \mathbf{G}_0 \mathbf{G}_\beta} \quad (15.23) - (15.24)$$

The ambiguity in sign in expression (15.23) is resolved by considering the sign of  $\frac{d^2\phi}{d\alpha^2}(\alpha_m)$ , we obtain the next relation:

$$\frac{d^2\phi(\alpha_m)}{d\alpha^2} = -\frac{2}{\beta}(\mathbf{G}_0 + \mathbf{Z}) + 2(\mathbf{G}_0 + \mathbf{G}_\beta + 2\mathbf{Z})\frac{\alpha_m}{\beta^2} = \pm 2\omega/\beta \quad (15.25)$$

When second derivate of  $\frac{d^2\phi(\alpha_m)}{d\alpha^2}$  is positive we have minimum. Then we must take “+” sign in expression (15.23)

$$\frac{\alpha_m}{\beta} = \frac{\mathbf{G}_0 + \mathbf{Z} + \omega}{\mathbf{G}_0 + \mathbf{G}_\beta + 2\mathbf{Z}} \quad (15.26)$$

Davidon found that for greater numerical accuracy it is preferable to use the equivalent formula

$$\frac{\alpha_m}{\beta} = 1 - \frac{\mathbf{G}_\beta + \omega - \mathbf{Z}}{\mathbf{G}_\beta - \mathbf{G}_0 + 2\omega} \quad (15.27)$$

Given a point  $\{\mathbf{x}_k\}$  and a direction of search  $\{\mathbf{S}\}$ , where  $\{\mathbf{S}\}$  need not be a unit vector, Davidon’s cubic interpolation method for minimizing the differentiable function  $f(\{\mathbf{x}\})$  on the line  $\{\mathbf{x}\} = \{\mathbf{x}_k\} + \alpha\{\mathbf{S}\}$  is as follows:

1. Evaluate  $\mathbf{f}_0 = f(\{\mathbf{x}\})$  and  $\mathbf{G}_0 = \left\{ \frac{\partial f(\{\mathbf{x}_k\})}{\partial \mathbf{x}} \right\}^T \{\mathbf{S}\}$ . Check that  $\mathbf{G}_0 < 0$ . Choose  $\chi = 2$  and  $\mathbf{f}_m$  and determine  $\beta$  (expression 15.14).
2. Evaluate  $\mathbf{f}_\beta = f(\{\mathbf{x}_k\} + \beta\{\mathbf{S}\})$  and  $\mathbf{G}_\beta = \{g(\{\mathbf{x}_k\} + \beta\{\mathbf{S}\})\}^T \{\mathbf{S}\}$ .
3. If  $\mathbf{G}_\beta > 0$ , or it  $\mathbf{f}_\beta > \mathbf{f}_0$  go to step 5, otherwise go to step 4.
4. Replace  $\beta$  by  $2\beta$  evaluate the new  $\mathbf{f}_\beta$  and  $\mathbf{G}_\beta$  and return to step 3.
5. Interpolate in the interval  $[0, \beta]$  for  $\alpha_m$  using expression (15.27).

Return to step 5 to repeat the interpolation in the smaller interval  $[0, \alpha_m]$  or  $[\alpha_m, \beta]$ , according to

$$| \{g(\{\mathbf{x}_k\} + \alpha_m)\{\mathbf{S}\}\}^T \{\mathbf{S}\} | < \epsilon, \epsilon > 0$$

1. Stop when the interval of interpolation has decreased to some prescribed value.

### 15.1.3.1. Input data for the solution of the problem

To solve the unconstrained optimization problem by Davidon cubic interpolation method, the *Cubic\_interpolation* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[\mathbf{i}])$  must be written in the procedure *F\_FUNK*. In the first line the number of variables (parameter

*nvar*) is coded. In the second line, a print code of intermediate results (*kprint*), step (*astep*) and a precision of the solution (*toler*) are coded. At reading of the information from the data file text lines are skipped. In each subsequent war lines, the number of the variable and the initial values of variable *X0* are coded.

The schematic structure of the datafile:

Text line \*  
*nvar*  
*kprint, astep, alfa, beta, gama, toler*  
 Text line \*  
 Vector *X0(nvar)*

### 15.1.3.2. Brief description of the Cubic\_interpolation program solving the optimization problem

The *Cubic\_interpolation* program was programmed on the *Maple*-language; it consists of the basic program and 6 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor are recorded into a file; therefore, a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.1.3.3. An example of use of the Maple-program Cubic\_interpolation

Use Davidon’s cubic interpolation method to minimize the objective function (Rosenbrock’s “banana function”):

$$\min f(\{x\}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := 100(u[2] - u[1]^2)^2 + (1 - u[1])^2;$$

The number of variables is *nvar* = 2; *toler* =  $10^{-9}$ ; *f<sub>m</sub>* = 5. Initial solution *x0*[1]=-1.0; *x0*[2]=0.0. The result is next: *x*[1]=0.56337; *x*[2]=0.312674; *min* *f*(*x*1, *x*2)=0.192866. The number of iteration is equal 11. The *Cubic\_interpolation* program is intended for the solution of unconstrained optimization problem with several variables. The source module of the program in *Maple*-language is represented in file *Mb12\_3\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_3.dat* and *Mb12\_3.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

## 15.2. Gradient methods for unconstrained optimization

### 15.2.1. Gradient method

*Gradient methods* for finding a local minimum or maximum of an unconstrained function *f*(*x*) are based untimely on the simple fact that *f*(*x*) decreases or increases in the direction *S* according

to the directional derivative  $\left\{ \frac{\partial f}{\partial \{x\}} \right\}^T \{S\}$  is negative or *Gradient methods* use information available by computing the gradient vector of *f*(*x*)

$$\mathbf{g}(\{\mathbf{x}\}) = \{\nabla f(\{\mathbf{x}\})\} = \left[ \frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right]^T \quad (15.28)$$

The gradient methods are defined by the iterative algorithm

$$\{\mathbf{x}_{k+1}\} = \{\mathbf{x}_k\} + \alpha_k \{\mathbf{S}_k\}. \quad (15.29)$$

Vector  $\{\mathbf{S}_k\}$  represents a direction of search and scalar  $\alpha_k$  determines how far we should step in this direction. The direction of  $\{\mathbf{g}(\{\mathbf{x}\})\}$  coincides with that of greatest rate of change of  $f(\{\mathbf{x}\})$ , and has that rate of change as its magnitude. The gradient is a vector that points in the direction of steepest ascent; therefore  $-\{\mathbf{g}(\{\mathbf{x}\})\}$  is the direction of steepest descent. Given point  $\{\mathbf{x}_k\}$ , the best direction to move to reduce the function value would seem to be the one in which the function decreases most rapidly, namely:

$$\{\mathbf{x}_{k+1}\} = \{\mathbf{x}_k\} + \alpha_k \{\mathbf{g}(\{\mathbf{x}_k\})\} = \{\mathbf{x}_k\} - \alpha_k \{\mathbf{g}_k\} \quad (15.30)$$

where  $\alpha_k$  - positive scalar minimizing  $f(\{\mathbf{x}_k\} - \alpha\{\mathbf{g}_k\})$ . From point  $\{\mathbf{x}_k\}$  we search along the direction of negative gradient  $\{\mathbf{g}_k\}$  to minimum point on this line. This *minimum point* is  $\{\mathbf{x}_{k+1}\}$ .

### 15.2.1.1. Input data for the solution of the problem

To solve the *unconstrained optimization problem* by the method of steepest descent, the *Gradient* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[i])$  must be written in procedure *F\_Funk*. In the first line, the number of variables (parameter *nvar*) is coded. In the second line, a print code of intermediate results (*kprint*), the number of parameters iteration (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of the variable and the initial values of variable **X0** are coded. When reading the information from the data file, text lines are skipped.

The schematic structure of the datafile:

Text line \*

*nvar*

*kprint, niter, toler*

Text line \*

Vector **X0**(*nvar*)

### 15.2.1.2. Brief description of the Gradient program solving the problem

The *Gradient* program was programmed in the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and the precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.2.1.3. An example of use of the Maple-program Gradient

The objective function is

$$\min f(\{\mathbf{x}\}) = (x_1 - 1)^4 + (2x_2 - x_1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (\text{u}[1] - 1)^4 + (2 * \text{u}[2] - \text{u}[1])^2;$$

The number of variables is  $\mathbf{nvar} = 2$ ; niter 200;  $\mathbf{toler} = 10^{-3}$ . Initial solution  $\mathbf{x0}[1] = -1.0$ ;  $\mathbf{x0}[2] = 2.5$ . The result is as follows:  $\mathbf{x}[1] = 1.061171$ ;  $\mathbf{x}[2] = 0.530695$ ;  $\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2) = 1.404984 \cdot 10^{-5}$ ;  $\left\| \frac{\partial \mathbf{f}(\mathbf{x}_1, \mathbf{x}_2)}{\partial \{\mathbf{x}\}} \right\| = 9.98 \cdot 10^{-4}$ . The number of iteration is equal to 183. The *Gradient* program solves nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is represented in file *Mb12\_4\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_4.dat* and *Mb12\_4.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

### 15.2.2. Fletcher-Reeves and Polak-Ribiere method

Fletcher and Reeves derived a simple recurrence formula, which produces a sequence of mutually conjugated directions. Consider the problem of minimizing the quadratic function

$$\mathbf{f}(\{\mathbf{x}\}) = \frac{1}{2} \{\mathbf{x}\}^T [\mathbf{H}]\{\mathbf{x}\} + \{\mathbf{b}\}^T \{\mathbf{x}\} + \mathbf{c} \quad (15.31)$$

where  $[\mathbf{H}]$  is positive definite and symmetric, by successive linear searches along mutually conjugate directions. The initial step is in the direction of steepest descent

$$\{\mathbf{s}_1\} = -\left\{ \frac{\partial \mathbf{f}}{\partial \{\mathbf{x}\}} \right\} = -\{\mathbf{q}_1\} \quad (15.32)$$

Subsequently, mutually conjugate directions are chosen so that

$$\{\mathbf{s}_{k+1}\} = -\{\mathbf{g}_{k+1}\} + \sum_{i=1}^k \beta_i \{\mathbf{s}_i\} \quad (k = 1, 2, \dots) \quad (15.33)$$

Where coefficients  $\beta_i$  shall be determined. It turns out that all coefficients except  $\beta_k$  are 0, given

$$\{\mathbf{s}_{k+1}\} = -\{\mathbf{g}_{k+1}\} + \beta_k \{\mathbf{s}_k\} \quad (k = 1, 2, \dots), \text{ where } \beta_k = \frac{\{\mathbf{g}_{k+1}\}^T \{\mathbf{g}_{k+1}\}}{\{\mathbf{g}_k\}^T \{\mathbf{g}_k\}} \quad (15.34 - 15.35)$$

The Fletcher and Reeves method will find minimum of a positive *quadratic* function of  $\mathbf{n}$  variables in, at most,  $\mathbf{n}$  iterations. When used on *non-quadratic* functions, the method is iterative. The complete algorithm is:

Given  $\{\mathbf{x}_0\}$ , compute  $\{\mathbf{g}_0\} = \left\{ \frac{\partial \mathbf{f}(\{\mathbf{x}_0\})}{\partial \mathbf{x}} \right\}$  and  $\{\mathbf{s}_0\} = -\{\mathbf{g}_0\}$ . For  $k=0..n-1$ ; set  $\{\mathbf{x}_{k+1}\} = \{\mathbf{x}_k\} + \alpha_k \{\mathbf{s}_k\}$

where  $\alpha_k$  minimizes  $\mathbf{f}(\{\mathbf{x}_k\} + \alpha \{\mathbf{s}_k\})$ ; compute  $\{\mathbf{g}_{k+1}\} = \left\{ \frac{\partial \mathbf{f}(\{\mathbf{x}_{k+1}\})}{\partial \{\mathbf{x}\}} \right\}$ ; unless  $k=n-1$ , set

$\{\mathbf{s}_{k+1}\} = -\{\mathbf{g}_{k+1}\} + \beta_k \{\mathbf{s}_k\}$  where  $\beta_k = \frac{\{\mathbf{g}_{k+1}\}^T \{\mathbf{g}_{k+1}\}}{\{\mathbf{g}_k\}^T \{\mathbf{g}_k\}}$ . Replace  $\{\mathbf{x}_0\}$  by  $\{\mathbf{x}_n\}$  and go back to step

1. Another important method of this type is the Polak-Ribiere method, where

$$\beta_k = \frac{\{\mathbf{g}_{k+1}\} - \{\mathbf{g}_k\}^T \{\mathbf{g}_{k+1}\}}{\{\mathbf{g}_k\}^T \{\mathbf{g}_k\}} \quad (15.36)$$

is used to determine  $\beta_k$ .

### 15.2.2.1. Input data for the solution of the problem

To solve the unconstrained optimization problem by Fletcher-Reeves and Polak-Ribiere method, the *Fletcher\_Reeves\_Polak\_Ribiere* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[i])$  must be written in procedure *F\_FUNK*. In the first line, the number of variables (parameter *nvar*) and parameter *kod* are coded. If Fletcher-Reeves method **kod=1** is used, for Polak-Ribiere method is **kod=2**. In the second line, a print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of the variable and the initial values of variable **X0** are coded. When reading the information from the data file text lines are skipped.

The schematic structure of the datafile:

Text line \*  
*nvar, kod*  
*kprint, niter, toler*  
 Text line \*  
 Vector **X0**(*nvar*)

### 15.2.2.2. Brief description of the Fletcher\_Reeves\_Polak\_Ribiere program solving the problem

The *Fletcher\_Reeves\_Polak\_Ribiere* program was programmed in the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and the precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore, a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.2.2.3. An example of use of the Maple-program Fletcher-Reeves-Polak-Ribiere

The objective function is

$$\min f(\mathbf{x}) = (x_1 - 1)^4 + (2x_2 - x_1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (\mathbf{u}[1] - 1)^4 + (2 * \mathbf{u}[2] - \mathbf{u}[1])^2$$

Fletcher-Reeves method. The number of variables is **nvar = 2 ; kod = 1 ; niter = 100 ; toler = 10<sup>-4</sup>**. Initial solution **x0[1]=-1.0; x0[2]=2.5**. The result as follows: **X[1]= 1.021168 ; X[2]= 0.510586 ;**

**minf(x<sub>1</sub>, x<sub>2</sub>) = 2.00 · 10<sup>-7</sup> ;  $\left\| \frac{\partial f(\mathbf{x}_1, \mathbf{x}_2)}{\partial \mathbf{x}} \right\| = 3.39371 \cdot 10^{-5}$** . The number of iteration is equal to 88. The

*Fletcher\_Reeves\_Polak\_Ribiere* program solves nonlinear unconstrained optimization problems.

Polak-Ribiere method. The number of variables **nvar = 2 ; kod = 2 ; niter = 100 ; toler = 10<sup>-4</sup>**. Initial solution **x0[1]=-1.0; x0[2]=2.5**. The result as follows: **X[1]= 1.021168 ; X[2]= 0.510586 ;**

**minf(x<sub>1</sub>, x<sub>2</sub>) = 2.007 · 10<sup>-7</sup> ;  $\left\| \frac{\partial f(\mathbf{x}_1, \mathbf{x}_2)}{\partial \mathbf{x}} \right\| = 3.39371 \cdot 10^{-5}$** .

The source module of the program in *Maple*-language is represented in file **Mb12\_5\_new.mws** of the PROG\_EXE directory, while initial data for the test example, and also outcomes of its solution

are represented in files **Mb12\_5.dat** and **Mb12\_5.rez** accordingly of the PROBLEMS directory of archive attached to the present book.

### 15.2.3. Newton's methods

We described in the preceding subsections direct-search methods in which no derivatives were used and gradient methods, which use first derivatives  $\mathbf{[H]}$  consider the Taylor series expansion of  $\mathbf{f}(\{\mathbf{X}_k\})$  about point  $\{\mathbf{X}_k\}$ , the k-th approximation to the minimum point. The expansion up to the quadratic term is

$$\mathbf{f}(\{\mathbf{X}\}) \cong \mathbf{f}(\{\mathbf{X}_k\}) + \{\mathbf{g}_k\}^T (\{\mathbf{X}\} - \{\mathbf{X}_k\}) + \frac{1}{2} (\{\mathbf{X}\} - \{\mathbf{X}_k\})^T [\mathbf{H}_k] (\{\mathbf{X}\} - \{\mathbf{X}_k\}), \quad (15.37)$$

where  $\{\mathbf{g}_k\}$  is gradient vector, at point  $\{\mathbf{X}_k\}$ ;  $[\mathbf{H}_k] = \begin{bmatrix} \frac{\partial^2 f_k}{\partial x_1^2} & \dots & \frac{\partial^2 f_k}{\partial x_1 \partial x_n} \\ \dots & \dots & \dots \\ \frac{\partial^2 f_k}{\partial x_n \partial x_1} & \dots & \frac{\partial^2 f_k}{\partial x_n^2} \end{bmatrix}$ ;  $f_k = f(\{\mathbf{x}_k\})$ .

At the optimum, the condition  $\{\mathbf{g}(\{\mathbf{X}\})\} = \mathbf{0}$  must be satisfied. Differentiating (15.37), we can find vector  $\{\mathbf{X}\}$ , which satisfies the above condition from

$$\{\mathbf{g}_k\} + [\mathbf{H}_k] (\{\mathbf{X}\} - \{\mathbf{X}_k\}) = \mathbf{0} \quad (15.38)$$

which can also be written

$$[\mathbf{H}_k] \{\mathbf{X}\} = [\mathbf{H}_k] \{\mathbf{X}_k\} - \{\mathbf{g}_k\} \quad (15.39)$$

Premultiplying by  $[\mathbf{H}]^{-1}$  and designating  $\{\mathbf{X}\}$  as the new value  $\{\mathbf{X}_{k+1}\}$ , we obtain

$$\{\mathbf{X}_{k+1}\} = \{\mathbf{X}_k\} - [\mathbf{H}_k]^{-1} \{\mathbf{g}_k\} \quad (15.40)$$

This is the traditional Newton-Raphson method. In view of the second-order sufficiency conditions for a minimum point, we assume that at a relative minimum point,  $\{\mathbf{X}^*\}$ , the Hessian matrix  $[\mathbf{H}(\{\mathbf{X}^*\})]$  is definitely positive. We can then argue that if  $\mathbf{f}(\{\mathbf{X}\})$  has continuous second partial derivatives,  $[\mathbf{H}(\{\mathbf{X}\})]$  is positive definite near  $\{\mathbf{X}^*\}$  and hence the method is well defined near the solution. Although Newton's method is very attractive in terms of its convergence properties near the solution, it requires modification before it the solution. The first modification is that usually a search parameter  $\alpha$  is introduced, so the method takes the following form:

$$\{\mathbf{X}_{k+1}\} = \{\mathbf{X}_k\} + \alpha_k \{\mathbf{S}_k\} \quad (15.41 - 15.43)$$

where  $\{\mathbf{S}_k\} = -[\mathbf{H}_k]^{-1} \{\mathbf{g}_k\}$  and  $\alpha_k$  is selected to minimize  $\min_{\alpha} \mathbf{f}(\{\mathbf{X}_k\} + \alpha \{\mathbf{S}_k\})$ . This is generalized Newton method. Upon the solution we expect, on the basis of how Newton's method was derived that  $\alpha_k \cong 1$ . Introducing the parameter for general points, however, guards against the possibility that the objective function increases with  $\alpha_k = 1$  to no quadratic terms in the objective function. The basic considerations required for developing the second modification can be seen most clearly by brief examination of the general class of algorithms

$$\{\mathbf{X}_{k+1}\} = \{\mathbf{X}_k\} - \alpha_k [\mathbf{B}_k] \{\mathbf{g}_k\} \quad (15.44)$$

where  $[B_k]$  is quadratic matrix,  $\alpha_k$  is a positive search parameter, and  $\{g_k\} = \left\{ \frac{\partial f(\{X_k\})}{\partial \{X\}} \right\}$ . We note that both steepest descent ( $[B_k] = [E]$ ) is a unit matrix, and Newton's method ( $[B_k] = [H_k^{-1}]$ ) belongs to this class. The direction vector  $\{s_k\} = -[B_k]\{g_k\}$  obtained in this way is a direction of descent for small  $\alpha_k$ , the value of  $f(\{X\})$  decreases as  $\alpha_k$  increases from zero. For small  $\alpha_k$ , we say

$$f(\{X_{k+1}\}) = f(\{X_k\}) + \{g_k\}^T (\{X_{k+1}\} - \{X_k\}) + o(\|\{X_{k+1}\} - \{X_k\}\|^2) \quad (15.45)$$

Using (15.44) this can be written as follows:

$$f(\{X_{k+1}\}) = f(\{X_k\}) - \alpha_k \{g_k\}^T [B_k] \{g_k\} + o(\alpha_k^2) \quad (15.46)$$

Hence if one is to guarantee a decrease in  $f(\{X\})$  for small  $\alpha_k$ , we must have  $\{g_k\}^T [B_k] \{g_k\} > 0$ , and matrix  $[G_k]$  is definitely positive. Setting  $[B_k] = [E]$  (unit matrix) to obtain the method of steepest descent, is about the simplest way to guarantee descent, but this method converges only linearly. Setting  $[B_k] = [H_k]^{-1}$  yields rapid descent near the solution but for a general point it may not yield a direction of descent, since  $[H_k]^{-1}$  may not be definitely positive or even may not exist. In practice, Newton's method must be modified to accommodate the possible no positive definiteness at regions remote from the solution. A common approach is to take  $[B_k] = (\lambda_k [E] + [H_k])^{-1}$  for some nonnegative value  $\lambda_k$ . This can be regarded as a kind of compromise between steepest descent ( $\lambda_k$  very large,  $\alpha_k = 10^4$ ) and Newton's method ( $\alpha_k = 0$ ). There is always an  $\lambda_k$  that makes  $[B_k]$  positive definite. The second modification that the method takes the following form:

$$\{X_{k+1}\} = \{X_k\} - \alpha_k (\lambda_k [E] + [H_k])^{-1} \{g_k\} \quad (15.47)$$

is called Marquardt method [226, 227].

### 15.2.3.1. Input data for the solution of the problem

To solve the unconstrained optimization problem by Newton's methods the *Newtons\_method* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(u[i])$  must be written in the procedure *F\_FUNK*. In the first line, the number of variables (parameter *nvar*) and *kod* (*kod*=1 if used Newton method, *kod*=2 - generalized Newton method and *kod*=3 - Marquardt method) are coded. In the second line, a print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of variable and initial values of variable **X0** are coded. When reading the information from the data file, text lines are skipped.

The schematic structure of the datafile:

Text line \*  
*nvar, kod*  
*kprint, niter, toler*  
 Text line \*  
 Vector **X0**(*nvar*)

### 15.2.3.2. Brief description of the `Newton's_method` program solving the optimization problem

The `Newton's_method` program was programmed in the *Maple*-language; it consists of the basic program and procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and the precision of the solution. The calculation results are output on the monitor and are recorded into a file; a qualifier of a target file must be ascribed to variable `file_rez1` of the program.

### 15.2.3.3. An example of use of the *Maple*-program `Newton's_method`

The objective function is

$$\min f(\{x\}) = (x_1 - 1)^4 + (2x_2 - x_1)^2$$

This objective function need write in *Maple* procedure `F_FUNK` the next form:

$$\text{rez} := (u[1] - 1)^4 + (2 * u[2] - u[1])^2$$

The number of variables is `nvar = 2`; `toler = 10-6`. Initial solution `X0[1] = -1.0`; `X0[2] = -2.50`. The result as follows:

Newton method: `kod = 1`; `X[1] = 0.984585`; `X[2] = 0.492293`; `minf(x1, x2) = 5.64598 · 10-8`. The number of iterations is 12, solution time is 1.01 sec;

Generalized Newton method: `kod=2`; `x[1]= 0.990162`; `x[2]=0.495053`; `minf(x1, x2) = 1.236895 · 10-8`. The number of iterations is 9, solution time is 3.5 sec.

Marquardt method: `kod = 3`; `X[1] = 0.99667`; `X[2] = 0.498335`; `minf(x1, x2) = 1.230301 · 10-10`. The number of iterations is 29, solution time is 0.8 sec.

The `Newton's_method` program solves nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is represented in file `Mb12_6_new.mws` of the `PROG_EXE` directory, while initial data for the test example, and also outcomes of its solution are represented in files `Mb12_6.dat` and `Mb12_6.rez` accordingly of the `PROBLEMS` directory of the archive attached to the present book.

## 15.3. Quasi-Newton methods

The assumption that evaluation and the use of Hessian matrix is impractical or costly, the idea underlying *quasi*-Newton methods is to use an approximation to the inverse Hessian in place of the true inverse that is required in Newton's method. The form of the approximation varies among different methods-ranging from the simplest where it remains fixed throughout the iterative process, to the more advanced where improved approximations are built up on the basis of information gathered during the descent process.

### 15.3.1. Simple rank procedure

The fundamental idea behind most *quasi*-Newton methods is to try to construct the inverse Hessian or an approximation of it, using information gathered as the descent process progress. The current approximation  $[B_k]$  is then used at each stage to define the next descent direction by setting

$[\mathbf{H}_k]^{-1} = [\mathbf{B}_k]$  in the modified Newton method. The inverse Hessian matrix can be built using gradient information obtained at various points.

Let as considered objective function  $f(\{\mathbf{X}\})$  has continuous second partial derivatives. If for two points  $\{\mathbf{X}_k\}$ ,  $\{\mathbf{X}_{k+1}\}$  we define  $\mathbf{g}_{k+1} = \left\{ \frac{\partial f(\{\mathbf{X}_{k+1}\})}{\partial \mathbf{x}} \right\}$ ,  $\mathbf{g}_k = \left\{ \frac{\partial f(\{\mathbf{X}_k\})}{\partial \mathbf{x}} \right\}$  and  $\{\mathbf{p}_k\} = \{\mathbf{X}_{k+1}\} - \{\mathbf{X}_k\}$ , then

$$\{\mathbf{g}_{k+1}\} - \{\mathbf{g}_k\} \cong [\mathbf{H}_k] \{\mathbf{p}_k\} \quad (15.48)$$

and if  $[\mathbf{H}_k]$  is constant, then we have

$$\{\mathbf{q}_k\} \cong \{\mathbf{g}_{k+1}\} - \{\mathbf{g}_k\} = [\mathbf{H}] \{\mathbf{p}_k\} \quad (15.49)$$

From (15.49) the evaluation of the gradient at two points gives information about Hessian matrix  $[\mathbf{H}]$ . If  $n$  linearly independent directions  $\{\mathbf{p}_0\}, \{\mathbf{p}_1\}, \dots, \{\mathbf{p}_{n-1}\}$  and the corresponding  $\{\mathbf{q}_k\}$  are known the  $[\mathbf{H}]$  matrix is uniquely determined. If  $[\mathbf{P}]$  and  $[\mathbf{Q}]$  matrices with columns  $\{\mathbf{p}_k\}$  and  $\{\mathbf{q}_k\}$  ( $k = 1, \dots, n$ ) respectively, we have

$$[\mathbf{Q}] = [\mathbf{H}][\mathbf{P}] \quad (15.50)$$

and

$$[\mathbf{H}] = [\mathbf{Q}][\mathbf{P}]^{-1} \quad (15.51)$$

Then we can construct approximations  $[\mathbf{B}_k]$  to  $[\mathbf{H}]^{-1}$  based on the data obtained from the first  $k$  steps of a descent process in such a way that if  $[\mathbf{H}]$  were constant the approximation would be consistent with (15.49) for these steps. If  $[\mathbf{H}]$  were constant,  $[\mathbf{B}_{k+1}]$  would satisfy

$$[\mathbf{B}_{k+1}]\{\mathbf{q}_i\} = \{\mathbf{p}_i\}, \quad 0 \leq i \leq k \quad (15.52)$$

After  $n$  linearly independent steps we would then have  $[\mathbf{B}_n] = [\mathbf{H}]^{-1}$ . The Hessian matrix  $[\mathbf{H}]$  and its inverse matrix  $[\mathbf{B}] = [\mathbf{H}]^{-1}$  are symmetric matrices. It is natural to require that  $[\mathbf{B}_k]$ , the approximation to  $[\mathbf{H}]^{-1}$ , be symmetric. We investigate the possibility of the defining a recursion of the next form:

$$[\mathbf{B}_{k+1}] = [\mathbf{B}_k] + \mathbf{a}_k \{\mathbf{d}_k\} \{\mathbf{d}_k\}^T \quad (15.53)$$

which preserves symmetry. The vector  $\{\mathbf{d}_k\}$  and the constant  $\mathbf{a}_k$  define a matrix of rank one, by which the approximation to the inverse is updated. Setting  $\mathbf{I}$  equal to  $\mathbf{k}$  in (15.52) and substituting (15.53) we obtain

$$\{\mathbf{p}_k\} = [\mathbf{B}_{k+1}]\{\mathbf{q}_k\} = [\mathbf{B}_k]\{\mathbf{q}_k\} + \mathbf{a}_k \{\mathbf{d}_k\} \{\mathbf{d}_k\}^T \{\mathbf{q}_k\} \quad (15.54)$$

Taking the inner product with  $\{\mathbf{q}_k\}$  we obtain

$$\{\mathbf{q}_k\}^T \{\mathbf{p}_k\} - \{\mathbf{q}_k\}^T [\mathbf{B}_k] \{\mathbf{q}_k\} = \mathbf{a}_k \{\mathbf{q}_k\}^T \{\mathbf{d}_k\} \{\mathbf{d}_k\}^T \{\mathbf{q}_k\} \quad (15.55)$$

Using (15.53), (15.54) can be written as follows

$$[\mathbf{B}_{k+1}] = [\mathbf{B}_k] + \frac{(\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})(\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})^T}{\mathbf{a}_k \{\mathbf{q}_k\}^T \{\mathbf{d}_k\} \{\mathbf{d}_k\}^T \{\mathbf{q}_k\}} \quad (15.56)$$

and using (15.55), we obtain

$$[\mathbf{B}_{k+1}] = [\mathbf{B}_k] + \frac{(\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})(\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})^T}{\{\mathbf{q}_k\}^T (\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})} \quad (15.57)$$

We have determined what rank shall be corrected if it is to satisfy (15.52) for  $i = k$ . The procedure is: starting with any symmetric positive definite matrix  $[\mathbf{B}_0]$ , any point  $\{\mathbf{X}_0\}$ , and with  $\mathbf{k} = 0$ .

1. Set  $\{\mathbf{S}_k\} = -[\mathbf{B}_k]\{\mathbf{q}_k\}$ .
2. Minimize  $f(\{\mathbf{X}_k\} + \alpha\{\mathbf{S}_k\})$  with respect to  $\alpha \geq 0$  to obtain  $\{\mathbf{X}_{k+1}\} = \{\mathbf{X}_k\} + \alpha_k\{\mathbf{S}_k\}$ ,  $\{\mathbf{p}_k\} = \alpha_k\{\mathbf{S}_k\}$ , and  $\{\mathbf{g}_{k+1}\}$ .
3.  $[\mathbf{B}_{k+1}]$  calculate according to (15.57) update  $\mathbf{k}$  and return to step 1.

There are some difficulties with this simple rank procedure:

1. the updating formula (15.57) preserves positives definiteness only if  $\{\mathbf{q}_k\}^T (\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\}) > 0$ , which cannot be guaranteed.
2. if  $\{\mathbf{q}_k\}^T (\{\mathbf{p}_k\} - [\mathbf{B}_k]\{\mathbf{q}_k\})$  is positive, it may be small, which can lead to numerical difficulties.

### 15.3.1.1. Input data for the solution of the problem

To solve the unconstrained optimization problem by *simple rank method*, the *QUASI\_SIMPLE* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[\mathbf{i}])$  must be written in procedure *F\_FUNK*. In the first line, the number of variables (parameter *nvar*) is coded. In the second line, the print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (*toler*) are coded. When reading the information from the data file text lines are skipped. In each subsequent war lines, the number of the variable and initial values of variable **X0** are coded.

The schematic structure of the datafile:

Text line \*  
*nvar*  
*kprint, niter, toler*  
 Text line \*  
 Vector **X0**(*nvar*)

### 15.3.1.2. Brief description of the QUASI\_SIMPLE program solving the optimization problem

The *QUASI\_SIMPLE* program was programmed on the *Maple*-language; it consists of the basic program and 10 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.3.1.3. An example of use of the Maple-program QUASI\_SIMPLE

The objective function is equal to

$$\min f(\{\mathbf{x}\}) = x_1^2 - x_1x_2 + 3x_2^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* the next form:

$$\text{rez} := (u[1]^2 - u[1] * u[2] - 3 * u[2]^2)$$

The number of variables is  $nvar = 2$ ;  $niter = 100$ ;  $toler = 10^{-6}$ . Initial solution  $X0[1] = -1.0$ ;  $X0[2] = 2.5$ . The result as follows:  $X[1] = 0$ ;  $X[2] = -5,5511 \cdot 10^{-17}$ ;  $\min f(x_1, x_2) = 9,244 \cdot 10^{-33}$ . The number of iteration is equal to 2. *QUASI\_SIMPLE* program solves nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is represented in file *Mb12\_7\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_7.dat* and *Mb12\_7.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

### 15.3.2. Davidon-Fletcher-Powell method

One of the most clever schemes for constructing the inverse Hessian, was originally proposed by Davidon and later developed by Fletcher and Powell. It has a desirable property that, for a quadratic objective, it simultaneously generates the directions of the conjugate gradient method while constructing the inverse Hessian. At each step the inverse Hessian is updated by the sum of two symmetric ranks of matrices, and this scheme is therefore often applied as the variable metric method.

#### 15.3.2.1. Algorithm of Davidon-Fletcher-Powell method

The procedure is as follows: starting with any symmetric positive definite matrix  $[B_0]$ , any point  $\{X_0\}$ , and with  $k = 0$  [225-227].

1. Set  $\{S_k\} = -[B_k]\{g_k\}$ .
2. Minimize  $f(\{X_k\} + \alpha\{S_k\})$  with respect to  $\alpha \geq 0$  to obtain  $\{X_{k+1}\} = \{X_k\} + \alpha_k\{S_k\}$ ,  $\{p_k\} = \alpha_k\{S_k\}$  and  $\{g_{k+1}\}$ .
3. Set  $\{q_k\} = \{g_{k+1}\} - \{g_k\}$  and  $[B_{k+1}] = [B_k] + \frac{\{p_k\}\{p_k\}^T}{\{p_k\}^T\{q_k\}} - \frac{[B_k]\{q_k\}\{q_k\}^T[B_k]}{\{q_k\}^T[B_k]\{q_k\}}$ .

Update  $k$  and return to step 1.

#### 15.3.2.2. Input data for the solution of the problem

To solve the *unconstrained optimization problem* by Davidon-Fletcher-Powell method, the *Davidon\_Fletcher\_Powell* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable *F*. The data in the file are placed in the strict order, namely. The expression of the objective function  $f(u[i])$  must be written in the procedure *F\_FUNK*. In the first line, the number of variables (parameter *nvar*) is coded. In the second line, a print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of the variable and the initial values of variable *X0* are coded. When reading the information from the data file text lines are skipped.

The schematic structure of the datafile:

Text line \*

*nvar*

*kprint, niter, toler*

Text line \*  
Vector  $X0(nvar)$

### 15.3.2.3. Brief description of the Davidon\_Fletcher\_Powell program solving the problem

The *Davidon\_Fletcher\_Powell* program was programmed in the *Maple*-language; it consists of the basic program and 10 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.3.2.4. An example of use of the Maple-program Davidon\_Fletcher\_Powell

The objective function (Rosenbrock's "banana function") is

$$\min f(\{x\}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (100 * (u[2] - u[1]^2)^2 + (u[1] - 1)^2)$$

The number of variables is  $nvar = 2$ ;  $niter = 100$ ;  $toler = 10^{-6}$ . Initial solution  $x0[1] = -1.9$ ;  $x0[2] = 2$ . The result is next:  $X[1] = 1.0$ ;  $X[2] = 1.0$ ;  $\min f(x_1, x_2) = 4.687742 \cdot 10^{-15}$ . The number of iterations is equal to 16. The program *Davidon\_Fletcher\_Powell* solves nonlinear unconstrained optimization problems. The source module of the program in *Maple* is presented in file *Mb12\_8\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_8.dat* and *Mb12\_8.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

### 15.3.3. Broyden-Fletcher-Goldfarb-Shanno method

A Broyden method is defined as a *quasi*-Newton method in which at each iteration a member of the Broyden family is used as the updating formula. Broyden-Fletcher-Goldfarb-Shanno method is the Broyden *family method*.

#### 15.3.3.1. Algorithm of Broyden-Fletcher-Goldfarb-Shanno method

The procedure of this method is: starting with any symmetric positive definite matrix  $[B_0]$ , any point  $\{X_0\}$ , and with  $k = 0$  [227].

1. Set  $\{S_k\} = -[B_k]\{g_k\}$ .
2. Minimize  $f(\{X_k\} + \alpha\{S_k\})$  with respect to  $\alpha \geq 0$  to obtain  $\{X_{k+1}\} = \{X_k\} + \alpha_k\{S_k\}$ ,  $\{p_k\} = \alpha_k\{S_k\}$ , and  $\{g_{k+1}\}$ .
3. Set  $\{q_k\} = \{g_{k+1}\} - \{g_k\}$  and

$$[B_{k+1}] = [B_k] + \left( \frac{1 + \{q_k\}^T [B_k] \{p_k\}}{\{q_k\}^T \{p_k\}} \right) \frac{\{p_k\} \{p_k\}^T}{\{p_k\}^T \{q_k\}} - \frac{\{p_k\} \{q_k\}^T [B_k] + [B_k] \{q_k\} \{p_k\}^T}{\{q_k\}^T \{p_k\}}$$

Update  $k$  and return to step 1. The last formula an important update formula that can be used exactly like the Davidon-Fletcher-Powell formula.

### 15.3.3.2. Input data for the solution of the problem

To solve the *unconstrained optimization problem* by Broyden-Fletcher-Goldfarb-Shanno method, the *BFGS* program is used. All data necessary for operation of this program, should be before-hand recorded in a file, therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $f(\mathbf{u}[i])$  must be written in procedure *F\_FUNK*. In the first line the number of variables (parameter *nvar*) is coded. In the second line, a print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of variable and initial values of variable **X0** are coded.

The schematic structure of the datafile:

Text line \*

*nvar*

*kprint, niter, toler*

Text line \*

Vector **X0**(*nvar*)

### 15.3.3.3. Brief description of the BFGS program solving the optimization problem

The *BFGS* program was programmed in the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.3.3.4. An example of use of the Maple-program BFGS

The objective function (Rosenbrock's "banana function") is

$$\min f(\{x\}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (100 * (u[2] - u[1]^2)^2 + (u[2] - 1)^2);$$

The number of variables is **nvar = 2**; **niter = 100**; **toler = 10<sup>-6</sup>**. Initial solution **x0[1]=-1.9**; **x0[2]=2**.

The result as follows: **x[1]=1.0**; **x[2]=1.0**; **minf(x<sub>1</sub>, x<sub>2</sub>) = 4.69556 · 10<sup>-15</sup>**. The number of iteration is 16. *BFGS* program is solved nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is presented in file *Mb12\_9\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in datafiles **Mb12\_9.dat** and **Mb12\_9.rez** accordingly of the *PROBLEMS* directory of archive attached to the present book.

### 15.3.4. Memoryless quasi-Newton method

The preceding development of *quasi-Newton* methods can be used as a basis for reconsideration of conjugate gradient methods. The result is an attractive class of new procedures.

#### 15.3.4.1. Algorithm of memoryless quasi-Newton method

Consider a simplification of the Broyden-Fletcher-Goldfarb-Shanno (*BFGS*) *quasi-Newton* method where  $[B_{k+1}]$  is defined by a *BFGS* update applied to  $[B] = [E]$  (*unit matrix*), rather than to  $[B_k]$ .

Thus,  $[B_{k+1}]$  is determined without reference to the previous  $[B_k]$ ; hence, the update procedure is memoryless. This update procedure leads to the next algorithm [227]: start at any point  $\{x_0\}$ ,  $k = 0$ .

1. Set  $[B_k][E]$ .
2. Set  $\{S_k\} = -[B_k]\{g_k\}$ .
3. Minimize  $f(\{X_k\} + \alpha\{S_k\})$  with respect to  $\alpha \geq 0$  to obtain  $\alpha_k$ ,  $\{X_{k+1}\} = \{X_k\} + \alpha_k\{S_k\}$ ,  $\{p_k\} = \alpha_k\{S_k\}$ ,  $\{g_{k+1}\}$ ,  $\{q_k\} = \{g_{k+1}\} - \{g_k\}$ . (Select  $\alpha_k$  accurately enough to ensure  $\{p_k\}^T\{q_k\} > 0$ ).
4. If  $k$  is not an integer multiple of  $n$ , set.

$$[B_{k+1}] = [E] - \frac{\{q_k\}\{p_k\}^T + \{p_k\}\{q_k\}^T}{\{p_k\}^T\{q_k\}} + \left( \frac{1 + \{q_k\}^T\{q_k\}}{\{p_k\}^T\{q_k\}} \right) \frac{\{p_k\}\{p_k\}^T}{\{p_k\}^T\{q_k\}}$$

Add 1 to  $k$  and return to step 2. If  $k$  is an integer multiple of  $n$ , return to step 1.

### 15.3.4.2. Input data for the solution of the problem

To solve the unconstrained optimization problem by memoryless *quasi*-Newton method, the *Memoryless\_QN* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order. The expression of the objective function  $f(u[i])$  must be written in the procedure *F\_FUNK*. In the first line, the number of variables (parameter *nvar*) is coded. In the second line, a print code of intermediate results (*kprint*), the number of iterations (*niter*) and the precision of the solution (*toler*) are coded. In each subsequent war lines, the number of the variable and the initial values of variable  $X0$  are coded.

The schematic structure of the datafile:

Text line \*  
*nvar*  
*kprint, niter, toler*  
 Text line \*  
 Vector  $X0(nvar)$

### 15.3.4.3. Brief description of the Memoryless\_QN program solving the optimization problem

The *Memoryless\_QN* program was programmed on the *Maple*-language; it consists of the basic program and 5 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.3.4.4. An example of use of the Maple-program Memoryless\_QN

The objective function (Rosenbrock's "banana function") is

$$\min f(\{x\}) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := (100 * (u[2] - u[1]^2)^2 + (u[1] - 1)^2);$$

The number of variables is  $nvar = 2$ ;  $niter = 100$ ;  $toler = 10^{-6}$ . Initial solution  $x0[1]=-1.9$ ;  $x0[2]=2$ . The result as follows:  $\mathbf{X}[1] = 1.0$ ;  $\mathbf{X}[2] = 1.0$ ;  $\min f(\mathbf{x}_1, \mathbf{x}_2) = 7.202342 \cdot 10^{-15}$ . Number of iteration is equal to 21. *Memoryless\_QN* program solves nonlinear unconstrained optimization problems. The source module of the program in *Maple*-language is represented in file *Mb12\_10\_new.mws* of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files *Mb12\_10.dat* and *Mb12\_10.rez* accordingly of the *PROBLEMS* directory of archive attached to the present book.

## 15.4. Constrained minimization methods

Most problems in engineering optimization cannot be formulated as *unconstrained minimization problems*. The basic problem that we consider is the minimization of a function subject to inequality constraints

$$\min f(\{\mathbf{X}\}) \quad (15.58)$$

so, that

$$\mathbf{g}_j(\{\mathbf{X}\}) \leq 0, j = 1, \dots, m \quad (15.59)$$

or

$$\mathbf{g}_j(\{\mathbf{X}\}) \geq 0, j = 1, \dots, m \quad (15.60)$$

The *constraints* divide the design space into *two* domains, the feasible domain where the constraints are satisfied and the infeasible domain where at least one of the constraints is violated. A fundamental concept that provides a great deal of insight as well as simplifying the required theoretical development is that of an active constraint. An inequality constraint  $\mathbf{g}_j(\{\mathbf{X}\}) \leq 0$  is said to be active at a feasible point  $\{\mathbf{X}\}$  if  $\mathbf{g}_j(\{\mathbf{X}\}) = 0$  and inactive at  $\{\mathbf{X}\}$  if  $\mathbf{g}_j(\{\mathbf{X}\}) < 0$ .

### 15.4.1. Penalty-function method

The problem is to find  $\{\mathbf{X}\}$  so that (15.58). The idea behind penalty-function methods is simple. Rather than trying to solve the constrained problem, a penalty term that takes care of the constraints is added to the original objective function  $f\{\mathbf{X}\}$ . Penalty function  $\Psi(\{\mathbf{X}\}, \mathbf{r})$  is defined, and the problem is transformed to the minimization of

$$\Phi(\{\mathbf{X}\}, \mathbf{r}) = f(\{\mathbf{X}\}) + \mathbf{r} \sum_{j=1}^m \mathbf{G}(\mathbf{g}_j) \quad (15.61)$$

Function  $\mathbf{G}$  is selected so that, if minimization is performed for a sequence of values of  $\mathbf{r}$ , the solution will be forced to converge to that of the constrained problem. The factor  $\mathbf{r}$  performs the weighting between the objective function value and the penalty term, and it is often called the penalty parameter or response factor. The surfaces of  $\Phi(\{\mathbf{X}\}, \mathbf{r})$  are correspondingly termed response surfaces.

#### 15.4.1.1. Fiacco and McCormick method

There are two categories of penalty-function formulations:

1. Interior penalty (*barrier*)-function methods, in which all intermediate solutions lie intermediate solutions lie in the feasible region and converge to the solution from the interior side of the

acceptable. The advantage is that one may stop the search at any time and end up with a feasible and, hopefully, usable design. The penalty function of (15.61) can be defined

$$\Phi(\{\mathbf{X}\}, r) = f(\{\mathbf{X}\}) + r \sum_{j=1}^m \frac{1}{g_j(\{\mathbf{X}\})} \quad (15.62)$$

2. Exterior penalty-function methods, where all intermediate solutions lie in the feasible region solution from the outside. One advantage of this method is that the solution may be started from an infeasible point, eliminating the need for an initial feasible point.

The penalty function of (15.62) can be defined as follows:

$$\Phi(\{\mathbf{X}\}, r) = f(\{\mathbf{X}\}) + r \sum_{j=1}^m P_j^\gamma, \text{ where } P_j^\gamma = \begin{cases} g_j & \text{if } g_j > 0 \\ 0 & \text{if } g_j \leq 0 \end{cases} \quad (15.63) - (15.64)$$

and the exponent  $\gamma \geq 0$ .

Consider only interior penalty (*barrier*) function. The idea is to minimize (15.62) for a sequence of values of  $r$ , instead of solving the constrained problem of (15.60). Since, at an interior point, all the terms in the sum are negative, a positive choice of  $r$  will result in a positive penalty term to be added to  $f(\{\mathbf{X}\})$ . As a boundary of the feasible region is approached, some boundary of the feasible region approaches some  $g_j(\{\mathbf{X}\})$  will approach zero and the penalty term will approach.

Reducing recessively the parameter  $r$  is recommended to compute according formula [225,226]:

$$r = - \frac{\left\{ \frac{\partial f(\{\mathbf{X}_0\})}{\partial \{\mathbf{X}\}} \right\}^T \left\{ \frac{\partial G(\{\mathbf{X}_0\})}{\partial \{\mathbf{X}\}} \right\}}{\left\{ \frac{\partial G(\{\mathbf{X}_0\})}{\partial \{\mathbf{X}\}} \right\}^T \left\{ \frac{\partial G(\{\mathbf{X}_0\})}{\partial \{\mathbf{X}\}} \right\}} \quad (15.65)$$

#### 15.4.1.2. Input data for the solution of the problem

To solve the constrained optimization problem by Fiacco and McCormick method, the *Fiacco\_McCormick* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable  $\mathbf{F}$ . The data in the file are placed in the strict order, namely. The expression of the objective function  $f(\mathbf{u}[i])$  must be written in the procedure *F\_FUNK*. The expressions of the inequalities  $g_j(\{\mathbf{X}\}) \geq 0$  must be written in the procedure *F\_INEQUALITY* and each expression of inequality ascribes to elements of array *INEQ*. In the first line, the number of variables (parameter *nvar*) and the number of inequality (parameter *nbound*) are coded. In the second line, a print code of intermediate results (*kprint*), the number of iteration (*niter*) and the precision of the solution (parameters *toler1*, *toler2*). A simple criterion for checking convergence is computed

$$\frac{f_k - f_{k+1}}{f_k} < \text{toler1}$$

The parameter *toler2* is minimum value of  $r$ . In each subsequent war lines, the number of variable and initial values of variable  $\mathbf{X0}$  are coded.

The schematic structure of the datafile:

Text line \*  
*nvar, nbond*  
*kprint, niter, toler1, toler2*  
 Text line \*  
 Vector  $X0(nvar)$

**15.4.1.3. Brief description of the *Fiacco\_McCormick* program solving the optimization problem**

The *Fiacco\_McCormick* program was programmed in the *Maple*-language; it consists of the basic program and 12 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

**15.4.1.4. An example of use of the *Maple*-program *Fiacco\_McCormick***

*Fiacco\_McCormick* method is used to minimize the objective function

$$\min f(\{X\}) = 2(x_1 - 1) + (x_2 - 1)^2$$

with this system of inequalities:

$$\begin{aligned} g_1(X_1, X_2) &= 4 - x_1^2 - x_2^2 \geq 0, \\ g_2(X_1, X_2) &= 2x_2 - x_1 \geq 0, \\ g_3(X_1, X_2) &= 2x_1 - x_2 \geq 0, \\ g_4(X_1, X_2) &= x_1 \geq 0, \\ g_5(X_1, X_2) &= x_2 \geq 0. \end{aligned}$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := 2 * (u[1] - 1)^2 + (u[2] - 1)^2$$

The inequalities shall be written in *Maple* procedure *F\_INEQUALITY* in the following form:

$$\begin{aligned} \text{INEQ}[1] &:= 4 - u[1]^2 - u[2]^2; \text{INEQ}[2] := 2 * u[2] - u[1]; \text{INEQ}[3] := 2 * u[1] - u[2]; \text{INEQ}[4] := u[1]; \\ \text{INEQ}[5] &:= u[2] \end{aligned}$$

The *Fiacco\_McCormick* program solves nonlinear constrained optimization problems. The source module of the program in *Maple*-language is represented in datafile **Mb12\_11\_new.mws** of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files **Mb12\_11.dat** and **Mb12\_11.rez** accordingly of the *PROBLEMS* directory of archive attached to the present book.

**15.4.2. Complex method**

The *complex method* is called the modified Nelder and Mead method taking into account constraints.

### 15.4.2.1. Algorithm of Complex method

The solved optimization problem consists of minimization of objective function  $f\{\mathbf{X}\}$  where  $\{\mathbf{X}\}$  determined by explicit constraints [225, 226]:

$$l_j \leq X_j \leq U_j, j=1, \dots, m \quad (15.66)$$

and also implicit constraints

$$g_j(\{\mathbf{X}\}) \leq 0 \quad (15.67)$$

Where  $l_j$  and  $U_j$  are values of lower and top constraints of variable  $X_j$ . If objective function  $f\{\mathbf{X}\}$  and constraints  $g_j(\{\mathbf{X}\})$  are convex functions, then the problem has a unique solution. The initial solution  $\{\mathbf{X}_0\}$  should satisfy all constraints (15.66, 15.67). Other points, using Meld and Mead's method satisfying constraints (15.66), get out as follows:

$$X_{ij} = l_j + r(U_j - l_j) \quad (15.68)$$

Where  $k=2n$ ,  $i=2, \dots, 2n$ ,  $j=1, \dots$  and  $r$  is pseudorandom *uniformly* distributed variable in interval  $[0,1]$ .

### 15.4.2.2. Input data for the solution of the problem

To solve the constrained optimization problem by *Complex method*, the *Complex* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable  $F$ . The data in the file are placed in the strict order. The expression of the objective function  $f(u[i])$  must be written in the procedure  $F\_FUNK$ . The expressions of the inequalities  $g_j(\{\mathbf{X}\}) \geq 0$  must be written in the procedure  $F\_INEQUALITY$  and each expression of inequality ascribes to elements of array  $INEQ$ . In the first line, the number of variables (parameter  $nvar$ ) and the number of inequality (parameter  $nbound$ ) are coded. In the second line, a print code of intermediate results ( $kprint$ ), the number of iterations ( $niter$ ) and the precision of the solution (parameters  $toler1$ ,  $toler2$ ). A simple criterion for checking convergence is computed

$$\sqrt{\left( \sum_{i=1}^k f_i^2 - \left| \sum_{i=1}^k f_i \right|^2 / k \right) / k} \leq toler1, \quad \max \sqrt{\sum_{l=1}^n (X_{i,l} - X_{j,l})} \leq toler2$$

In each subsequent war lines, the number of the variable lower, initial and top values of variable are coded.

The schematic structure of the datafile:

Text line \*

$nvar, nbound$

$kprint, niter, toler1, toler2$

Text line \*

Vectors  $TL(nvar), X0(nvar), U(nvar)$

### 15.4.2.3. Brief description of the Complex program solving the optimization problem

The *Complex* program was programmed in the *Maple*-language; it consists of the basic program and 11 procedures. All procedures can be divided into two groups: procedures for data entry and for

calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

#### 15.4.2.4. An example of use of the *Maple*-program *Complex*

The *Complex method* is used to minimize the objective function

$$\min f(\{X\}) = x_1 - x_2 - 2x_3$$

with this system of inequalities:

$$\begin{aligned} g_1(X_1, X_2) &= 3x_1 + 4x_2 - 3x_3 - 23 \leq 0, \\ g_2(X_1, X_2) &= 5x_1 - 4x_2 - 3x_3 - 10 \leq 0, \\ g_3(X_1, X_2) &= 7x_1 + 4x_2 + 11x_3 - 30 \leq 0, \\ 0 &\leq x_1 \leq 10, \\ 0 &\leq x_2 \leq 10, \\ 0 &\leq x_3 \leq 10. \end{aligned}$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := u[1] - u[2] - 2 * u[3]$$

The inequalities shall be written in *Maple* procedure *F\_INEQUALITY* in the following form:

$$\begin{aligned} \text{INEQ}[1] &:= 3 * u[1] + 4 * u[2] - 3 * u[3] - 23; \quad \text{INEQ}[2] := 5 * u[1] - 4 * u[2] - 3 * u[3] - 10; \\ \text{INEQ}[3] &:= 7 * u[1] + 4 * u[2] + 11 * u[3] - 30 \end{aligned}$$

The number of variables is  $nvar = 3$ ; the number of inequalities is  $nbond = 3$ ; the number of iterations  $niter = 500$ ;  $toler1 = 10^{-10}$ ;  $toler2 = 10^{-5}$ . Initial solution  $X0[1] = 0.0$ ;  $X0[2] = 2.0$ ;  $X0[3] = 1.0$ .  $0 \leq X_1 \leq 10$ ;  $0 \leq X_2 \leq 0$ ;  $0 \leq X_3 \leq 10$ . The result as follows:  $X[1] = 0.000180$ ;  $X[2] = 6.124787$ ;  $X[3] = 0.4999$ ;  $\min f(X_1, X_2) = -7.124407$ ; the number of iteration is 306. The *Complex* program solves nonlinear constrained optimization problems. The source module of the program in *Maple*-language is represented in file **Mb12\_12\_new.mws** of the *PROG\_EXE* directory, while initial data for the test example, and also outcomes of its solution are represented in files **Mb12\_12.dat** and **Mb12\_12.rez** accordingly of the *PROBLEMS* directory of archive attached to the present book.

#### 15.4.3. Methods based on the sequence of linear programs. The first method

Consider the general nonlinear programming with inequality constraints

$$\min f(\{X\}) \tag{15.69}$$

and

$$g_j(\{X\}) \leq 0, j = 1, \dots, m \tag{15.70}$$

Sequential *linear programming* (LP) methods are based on successive linearizations of the *constraints* and the objective function. Taylor series expansion of  $f(\{X\})$  and  $g_j$  about point  $\{X_k\}$  up to linear terms are used. Then objective function (15.69) and constraints (15.70) can be approximated by

$$f(\{\mathbf{X}\}) \cong f(\{\mathbf{X}_k\}) + \left\{ \frac{\partial f(\{\mathbf{X}_k\})}{\partial \{\mathbf{X}\}} \right\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}) = \mathbf{f}_k + \{\mathbf{g}_k\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}), \quad (15.71) - (15.72)$$

$$\mathbf{g}_j(\{\mathbf{X}\}) \cong \mathbf{g}_j(\{\mathbf{X}_k\}) + \left\{ \frac{\partial \mathbf{g}_j(\{\mathbf{X}_k\})}{\partial \{\mathbf{X}\}} \right\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}) = \mathbf{g}_{jk} + \{\mathbf{G}_{jk}\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}), \quad j = 1, \dots, m.$$

The simplest approach for replacing the nonlinear programming problem by a sequence of the LP problems is to choose initial point  $\{\mathbf{X}_k\}$  and to find  $\{\mathbf{X}\}$  by solving

$$\min \mathbf{f}_k + \{\mathbf{g}_k\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}), \quad (15.73) - (15.74)$$

$$\mathbf{g}_{jk} + \{\mathbf{G}_{jk}\} (\{\mathbf{X}\} - \{\mathbf{X}_k\}) \leq \mathbf{0}, \text{ where subject to } \{\mathbf{X}\} \in \mathbf{S}, \mathbf{S} \subset \mathbf{E}^n \text{ is a closed convex set.}$$

This problem can be solved repeatedly, redefining  $\{\mathbf{X}_k\}$  each time as the optimal solution of the preceding problem.

### 15.4.3.1. Cutting plane method

The cutting-plane method is based on the useful property that linearized constraints is convex problems that always lie entirely outside the feasible region. If the solution is not sufficiently accurate, the binding constraints should be better presented by an additional number of approximating hypereplanes. The general form of a cutting-plane algorithm for problem (15.73, 15.74) is as follows:

Given a polytope  $\mathbf{P}_k \supset \mathbf{S}$ :

1. Minimize  $\mathbf{f}_k + \{\mathbf{g}_k\} (\{\mathbf{X}\} - \{\mathbf{X}_k\})$ , if  $\{\mathbf{X}_k\} \in \mathbf{S}$ , stop;  $\{\mathbf{X}_k\}$  is optimal, otherwise.
2. Final hyperplane  $\mathbf{H}_k$  separating point  $\{\mathbf{X}_k\}$  from  $\mathbf{S}$ , that is final  $\{\mathbf{a}_k\} \in \mathbf{E}^n$ ,  $\mathbf{b}_k \in \mathbf{E}^1$  such that  $\mathbf{S} \subset \{\{\mathbf{X}\} : \{\mathbf{a}_k\}^T \{\mathbf{X}\} \leq \mathbf{b}_k\}$ ,  $\{\mathbf{X}_k\} \in \{\{\mathbf{X}\} : \{\mathbf{a}_k\}^T \{\mathbf{X}\} > \mathbf{b}_k\}$ . Update  $\mathbf{P}_k$  to obtain  $\mathbf{P}_{k+1}$  including as a constraint  $\{\mathbf{a}_k\}^T \{\mathbf{X}\} \leq \mathbf{b}_k$ .

### 15.4.3.2. Input data for the solution of the problem

To solve the constrained optimization problem by *cutting plane method*, the *Cutting\_Plane* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable  $\mathbf{F}$ . The data in the file are placed in the strict order. The expression of objective function  $f(\mathbf{u}[i])$  must be written in procedure  $F\_FUNK$ . The expressions of inequalities  $\mathbf{g}_j(\{\mathbf{X}\}) \leq \mathbf{0}$  must be written in procedure  $F\_INEQUALITY$  and each expression of inequality ascribes to elements of array  $\mathbf{INEQ}$ . In the first line, the number of variables (parameter  $nvar$ ) and the number of inequality (parameter  $nbound$ ) are coded. In the second line, a print code of intermediate results ( $kprint$ ), the number of iterations ( $niter$ ) and the precision of the solution ( $toler$ ) are presented. In each subsequent war lines, the number of the variable, the initial value of variables  $\mathbf{X0}$  and lower values of variables  $\mathbf{Par}$ .

The schematic structure of the datafile:

Text line \*  
 $nvar, nbound$   
 $kprint, niter, toler$   
 Text line \*  
 Vectors  $\mathbf{X0}(nvar), \mathbf{Par}(nvar)$

### 15.4.3.3. Brief description of the Cutting\_Plane program solving the optimization problem

The *Cutting\_Plane* program was programmed in the *Maple*-language; it consists of the basic program and 15 procedures. All procedures can be divided into two groups: procedures for data entry and for calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.4.3.4. An example of use of the Maple-program Cutting\_Plane

The *Cutting plane method* is used to minimize the objective function

$$\min f(\{X\}) = 6.5x_6 - 0.5x_6^2 - x_1 - 2x_2 - 3x_3 - 2x_4 - x_5$$

with this system of inequalities:

$$\begin{aligned} g_1(x_1, \dots, x_6) &= x_6 - 2x_1^2 + 8x_2 + x_3 + 3x_4 + 5x_5 - 16 \leq 0, \\ g_2(x_1, \dots, x_6) &= -8x_6 - 4x_1 - 2x_2 + 2x_3 + 4x_4 - x_5 + 1 \leq 0, \\ g_3(x_1, \dots, x_6) &= 2x_6 + 0.5x_1 + 0.2x_2 - 3x_3 - x_4 - 4x_5 - 24 \leq 0, \\ g_4(x_1, \dots, x_6) &= 0.2x_6 + 2x_1 + 0.1x_2 - 4x_3 + 2x_4 + 2x_5 - 12 \leq 0, \\ g_5(x_1, \dots, x_6) &= -0.1x_6 - 0.5x_1 + 2x_2 + 5x_3 + 5x_4 + 3x_5 - 3 \leq 0, \\ g_6(x_1, \dots, x_6) &= x_3 - 1 \leq 0, \\ g_7(x_1, \dots, x_6) &= x_4 - 1 \leq 0, \\ g_8(x_1, \dots, x_6) &= x_5 - 2. \end{aligned}$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := 6.5 * u[6] - 0.5 * u[6]^2 - u[1] - 2 * u[2] - 3 * u[3] - 2 * u[4] - u[5]$$

The inequalities shall be written in *Maple* procedure *F\_INEQUALITY* in the following form:

$$\begin{aligned} \text{INEQ}[1] &:= u[6] - 2 * u[1]^2 + 8 * u[2] + u[3] + 3 * u[4] + 5 * u[5] - 16; \\ \text{INEQ}[2] &:= -8 * u[6] - 4 * u[1] - 2 * u[2] + 2 * u[3] + 4 * u[4] - u[5] + 1; \\ \text{INEQ}[3] &:= 2 * u[6] + 0.5 * u[1] + 0.2 * u[2] - 3 * u[3] - u[4] - 4 * u[5] - 24; \\ \text{INEQ}[4] &:= 0.2 * u[6] + 2 * u[1] + 0.1 * u[2] - 4 * u[3] + 2 * u[4] + 2 * u[5] - 12; \\ \text{INEQ}[5] &:= -0.1 * u[6] - 0.5 * u[1] + 2 * u[2] + 5 * u[3] - 5 * u[4] + 3 * u[5] - 3; \text{INEQ}[6] := u[3] - 1; \\ \text{INEQ}[7] &:= u[4] - 1; \text{INEQ}[8] := u[5] - 2 \end{aligned}$$

The number of variables is **nvar = 6**; the number of inequalities is **nbond = 8**; the number of iteration **niter = 20**; **toler1 = 10<sup>-9</sup>**. Initial solution **X0[1] = 0.1**; **X0[2] = 0.1**; **X0[3] = 0.1**; **X0[4] = 0.1**; **X0[5] = 0.1**; **X0[6] = 0.1**. The result as follows: **X[1] = 1.171573**; **X[2] = 1.156854**; **X[3] = 1**; **X[4] = 1**; **X[5] = 1.156854 · 10<sup>-32</sup>**; **X[6] = 0**; **minf(X<sub>1</sub>, ..., X<sub>6</sub>) = -8.485281**; the number of iteration is to 7. The *Cutting plane* program solves nonlinear constrained optimization problems. The source module of the program in *Maple*-language is represented in file **Mb12\_13\_new.mws** of the **PROG\_EXE** directory, while initial data for the test example, and also outcomes of its solution are represented in files **Mb12\_13.dat** and **Mb12\_13.rez** accordingly of the **PROBLEMS** directory of archive attached to the present book.

#### 15.4.4. The method based on the sequence of linear programs. The second method

The Griffith and Stewartore developed the method of approximate programming. The objective function and constraints are linearized by taking the first terms of the Taylor expansion about current point  $\{\mathbf{x}_k\}$ . However, we relinearize all the nonlinear relations at each iteration and no part of the preceding linear programming problem is retained. The *nonlinear programming (NLP)* is locally approximated by linear terms, permitting the solution of nonconvex programming problems. To ensure that the approximation is adequate, we limit the variation of  $\{\mathbf{x}\}$  in a typical linearized computation

$$|\mathbf{x}_{j,k+1} - \mathbf{x}_{j,k}| \leq \Delta_{j,k}, \quad (j = 1, \dots, n), \text{ where } \Delta_{j,k} \text{ is called move limits} \quad (15.75)$$

##### 15.4.4.1. Method of approximate programming

To solve a problem we first choose a starting point  $\{\mathbf{x}_k\}$  and linearize the objective function and the constraints in the neighborhood of  $\{\mathbf{x}_k\}$  (15.71 and 15.72). We solve the linearized problem of (15.73, 15.74) and redefine  $\{\mathbf{x}_k\}$  as the optimum solution to the preceding problem. The nonlinear functions are relimarized about new  $\{\mathbf{x}_k\}$ , and the process is repeated until either no significant improvement occurs in the solution or successive solutions start to oscillate between the vertices of the feasible region. In the latter event we reduce the values of bounds  $\Delta_{j,k}$ :

$$\Delta_{j,k+1} = \lambda \Delta_{j,k}, \text{ where } \lambda \text{ is parameter } 0 < \lambda < 1 \quad (15.76)$$

The general form of method of approximate programming algorithm for problem (15.73, 15.74) is as follows:

1. Given initial point  $\{\mathbf{x}_k\}$ ,  $\mathbf{k} = 0$ ,  $\{\Delta_k\}$ .
2. Linearize functions (15.71, 15.72) about point  $\{\mathbf{x}_k\}$ .
3. Minimize  $\mathbf{f}_k + \{\mathbf{g}_k\}(\{\mathbf{x}\} - \{\mathbf{x}_k\})$ .
4. Reduce parameter  $\lambda$  (15.76).
5. Reduce limits of  $\{\mathbf{x}\}$  (15.75).
6. Update  $\{\mathbf{x}_k\}$  to  $\{\mathbf{x}_{k+1}\}$ .

##### 15.4.4.2. Input data for the solution of the problem

To solve the constrained optimization problem by the method of *approximate programming method*, the *SLP* program is used. All data necessary for operation of this program, should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of the objective function  $\mathbf{f}(\mathbf{u}[\mathbf{i}])$  must be written in procedure *F\_FUNK*. The expressions of the inequalities  $\mathbf{g}_j(\{\mathbf{x}\}) \leq \mathbf{0}$  must be written in procedure *F\_INEQUALITY* and each expression of inequality ascribes to elements of array **INEQ**. In the first line, the number of variables (parameter *nvar*) and the number of inequality (parameter *nbound*) are coded. In the second line, a print code of intermediate results (*kprint*), maximum number of iteration (*niter*) and the precision of the solution (*toler*). In each subsequent war lines, the number of the variable and the initial values of variable **X0**, initial limits of variables **Par** and step of limits **DX** are coded.

The schematic structure of the datafile:

Text line \*  
*nvar, nbound*

*kprint, niter, toler*  
Text line \*  
Vectors  $X0(nvar)$ ,  $Par(nvar)$ ,  $DX(nvar)$

### 15.4.4.3. Brief description of the SLP program solving the optimization problem

The *SLP* program was programmed on the *Maple*-language; it consists of the basic program and 21 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

### 15.4.4.4. An example of use of the *Maple*-program SLP

The method of *approximate programming* is used to minimize the objective function

$$\min f(\{X\}) = x_1^2 + x_2^2 - 16x_1 - 10x_2$$

with this system of inequalities:

$$g_1(x_1, x_2) = -(11 - x_1^2 + 6x_1 - 4x_2) \leq 0,$$

$$g_2(x_1, x_2) = -(x_1 * x_2 - 3x_2 - \exp(x_1 - 3) + 1) \leq 0.$$

This objective function shall be written in *Maple* procedure *F\_FUNK* in the following form:

$$\text{rez} := u[1]^2 + u[2]^2 - 16 * u[1] - 10 * u[2];$$

The inequalities shall be written in *Maple* procedure *F\_INEQUALITY* the next form:

$$\text{INEQ}[1] := -11 + u[1]^2 - 6 * u[1] + 4 * u[2]; \quad \text{INEQ}[2] := -u[1] * u[2] + 3 * u[2] + \exp(u[1] - 3) - 1;$$

The number of variables is  $nvar = 2$ ; the number of inequalities is  $nbond = 2$ ; the number of iterations  $niter = 100$ ;  $toler = 10^{-9}$ . Initial solution  $X0[1] = 4.0$ ;  $X0[2] = 3.0$ . Lower and top limits of variable:  $Par[1,1] = 3.0$ ;  $Par[1,2] = 6.0$ ;  $Par[2,1] = 2.0$ ;  $Par[2,2] = 5.0$ . Maximal move limits of variables  $DX[1] = 0,510$ ;  $DX[2] = 0,70$ . The values of inequalities:

$$g_1(X_1, X_2) = -4.495542 \cdot 10^{-17};$$

$$g_2(X_1, X_2) = -4.137464 \cdot 10^{-15}.$$

The result as follows:  $X[1] = 5.239609$ ;  $X[2] = 3.746038$ ;  $\min f(X_1, X_2) = -79.807821$ ; the number of iterations  $iter = 37$ . The *SLP* program solves nonlinear constrained optimization problems. The source module of the program in *Maple*-language is represented in datafile **Mb12\_14\_new.mws** of the PROG\_EXE directory, while initial data for the test example, and also outcomes of its solution are represented in files **Mb12\_14.dat** and **Mb12\_14.rez** accordingly of the PROBLEMS directory of archive attached to the present book.

## 15.5. Genetic algorithms

*Genetic algorithms* (GA) have been the subject of considerable interest in recent years since they appear to provide a robust search procedure for solving difficult problems. GA are based on ideas from the science of genetics and the process of natural selection. GAs are powerful stochastic optimization methods based on the analogy with the mechanics of biological genetics and imitate the Darwinian survival of the fittest approach [230, 231].

Genetic algorithms search algorithms for finding optimal or near optimal solutions. They can be considered as a cross between gradient - based calculus methods and artificial intelligence algorithmic search methods. GAs use a directed search like gradient - based calculus methods but don't rely on derivatives. Thus they don't require the parameter space to be continuous. Along with that, they are global search methods. The basic genetic algorithm utilizes three genetic operations: reproduction (selection), crossover and mutation.

The GA works with an initial population which may correspond to numerical values of a particular variable. The size of this population may vary and is generally related to the problem under consideration. The members of this population are usually strings of zero and ones, i.e. binary strings

```

0 1 1 0 1 0 1 0
1 0 1 1 1 0 1 1
0 0 1 0 1 1 1 0
1 1 0 0 1 1 0 1
    
```

The strings themselves may be the encoded values of a variable or variables that we are examining. The initial population is generated randomly and we can use the terminology of genetic to characterize it. Each string in the population corresponds to a chromosome and each binary element of the string to a gene. The following optimization problem is solved

$$\min f(\{X\}) \tag{15.77}$$

and

$$l_i \leq x_i \leq u_i; \quad i = 1, \dots, n \tag{15.78}$$

where  $l_i$  and  $u_i$  lowest and highest limits of variable  $x_i$  correspondly. The number genes of each chromosome is equal to:

$$m_i \geq \log_2(N \cdot D_i + 1) \tag{15.79}$$

where  $D_i = u_i - l_i$  is length of interval;  $N$  - large number which specifies on how many parts divides interval  $D_i$ . The structure of the chromosome is made as follows: the first  $m_1$  a genes belong to variable  $x_1$ , from  $m_1 + 1$  to  $m_1 + m_2$  genes belong  $x_2$  and so on. The value of each variable  $x_i$  is defined as follows:

$$x_i = l_i + \text{decimal}(10101\dots01_2) D_i / 2^{m_i} - 1 \tag{15.80}$$

where decimal  $(10101\dots01_2)$  - value of a string in the decimal system. An initial population is generated entirely by randomly. The quality of a chromosome is measured by the objective is function

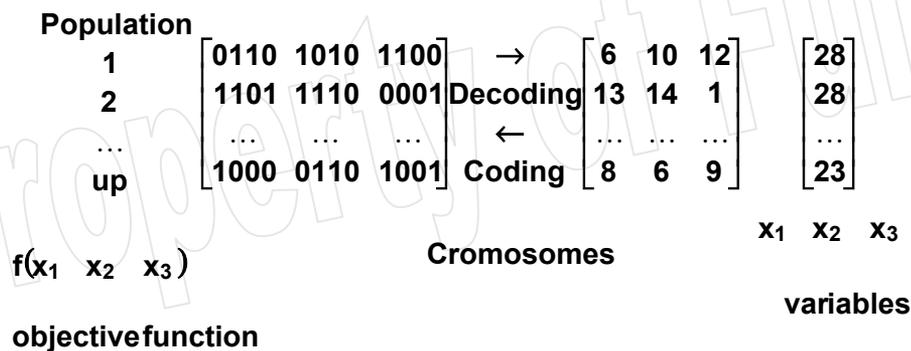
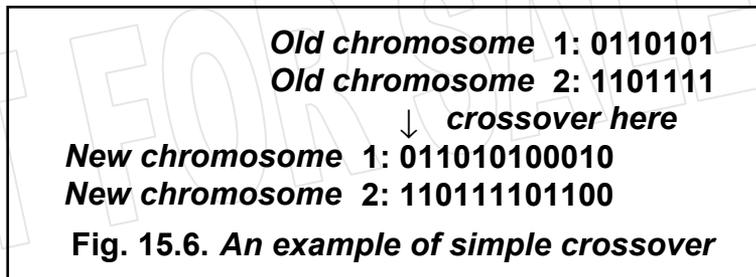
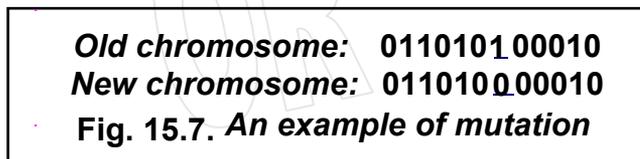


Fig. 15.5 Genetic algorithm as an optimization method  
 $(f(x_1, x_2, x_3) = x_1 + x_2 + x_3)$

*Selection* or *reproduction* is used to build the next generation using the objective function value of the chromosomes as their probability of survival. The more suitable an individual is the more chance it has to be a part of the next generation. The *crossover* operator picks two, parent chromosomes randomly from the new generation chooses a random crossover point and swags the chromosome strings after the crossover point strings after the crossover point (fig. 15.6). This way the operator produces two new chromosomes.



Then, a mutation operator is applied with a certain probability with changes of one or more elements of the chromosomes that are re-evaluated (fig. 15.7). This process is repeated for a fixed number of generation after which the chromosome with the best objective function value is considered to be the solution.



### 15.5.1. The procedure of genetic algorithm

The general procedure of GA is as follows:

1. Generate initial population **Pop<sub>k</sub>**, **k = 0**.
2. Evaluate value of objective function for each chromosome.
3. Generate new population **Pop<sub>k+1</sub>** using values of objective function
4. Crossover of **Pop<sub>k+1</sub>**.
5. Mutate genes of **Pop<sub>k+1</sub>**.
6. Update **Pop<sub>k+1</sub>** to **Pop<sub>k</sub>** and go to step two.

### 15.5.2. Input data for the solution of the problem

To solve the optimization problem by *genetic algorithm*, the *Genetic\_dalia* program is used. All data necessary for operation of this program should be before-hand recorded in a file; therefore the necessary qualifier of the file is ascribed to variable **F**. The data in the file are placed in the strict order. The expression of objective function **f(u[i])** must be written in procedure **F\_FUNK**. In the first line, the number of variables (parameter *nvar*), the number of *population (chromosomes)* (parameter *npopsiz*), the number of generation population (parameter *nkartu*), a large number which specifies on how many parts devices interval **D<sub>i</sub>** (parameter *ninterval*) are coded. In the second line, a print code of intermediate results (*kprint*), probability of crossover (*pkryz*), probability of imitation (*pmut*). In each subsequent war lines, the number of the variable, and the lowest and highest limits of variable **Par** are coded.

The schematic structure of the datafile:

Text line \*  
*nvar, nposize, nkartu, ninterval*  
*kprint, pkryz, pmut*  
 Text line \*  
 Matrix **Par**(*nvar*, 2)

**15.5.3. Brief description of the Genetic\_dalia program solving the optimization problem**

The *Genetic\_dalia* program was programmed in the *Maple*-language; it consists of the basic program and 24 procedures. All procedures can be divided into two groups: procedures for data entry and calculation. Time of its solution depends on the used number of variables and a precision of the solution. The calculation results are output on the monitor and are recorded into a file; therefore a qualifier of a target file must be ascribed to variable *file\_rez1* of the program.

**15.5.4. An example of use of the Maple-program Genetic\_dalia**

The objective function is

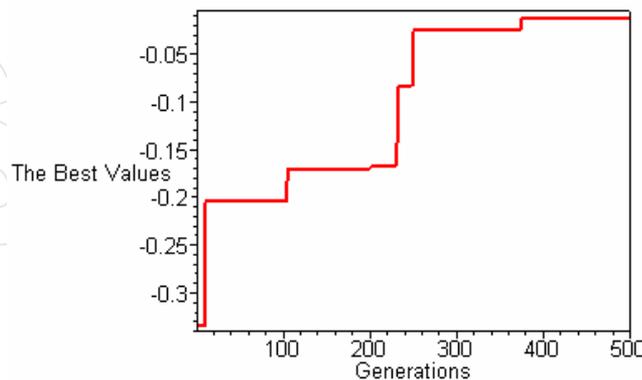
$$\min f(\{X\}) = -((x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 - 3)^2); \quad 0 \leq x_1 \leq 3, 0 \leq x_2 \leq 5, 0 \leq x_3 \leq 5, 0$$

This objective function shall be written in *Maple* procedure *F\_FUNK* the next form:

$$-((u[1]-1)^2 + (u[2]-2)^2 + (u[3]-3)^2)$$

The number of variables is **nvar = 3**; **nkartu = 500**; **interval = 10<sup>4</sup>**; **pkryz = 0.25**; **pmut = 0.085**.  
 Result is as follows **x[1]=1.068636**; **x[2]=2.081788**; **x[3]=2.977035**; **min f(x1,x2,x3)=-0.011928** (fig. 15.8).

*Objective Function*



**Fig. 15.8.** The Factor of safety versus number of generations (for example)

The *Genetic\_dalia* program solves nonlinear optimization problem. The source module of the program in *Maple*-language is represented in file **Mb12\_15\_new.mws** of the PROG\_EXE directory, while initial data for the test example, and also outcomes of its solution are represented in files **Mb12\_15.dat** and **Mb12\_15.rez** accordingly of the PROBLEMS directory of archive attached to the present book.

# References

1. **Aladjev V.Z.** *To the Theory of the Homogeneous Structures.*- Tallinn: Estonian Academy Press, 1972
2. **Aladjev V.Z., Osipov O.B.** *Introduction to Operating System of IBM/360 Computers, Vol. 1-2.*- Tallinn: CPTI V/O Sojuztorgsystem Press, 1975, 276 p. (in Russian with extended English summary)
3. **Aladjev V.Z., Osipov O.B.** *Introduction to Architecture of Computers IBM/360.*- Tallinn: Valgus Press, 1976, 332 p. (in Russian with extended English summary)
4. **Aladjev V.Z.** *Mathematical Theory of Homogeneous Structures and Their Applications.*- Tallinn: Valgus Press, 1980, 267 p.
5. **Aladjev V.Z. et al.** *Mathematical Developmental Biology.*- Moscow: Science Press, 1982, 254 p.
6. **Aladjev V.Z.** *Architecture and Software of PDP-11 Computers.*- Tallinn: IVC Gosbanka USSR, 1983
7. **Aladjev V.Z.** *Lecture on Personal Computer WANG 2200MVP.*- Tallinn: SKB MPCM ESSR, 1986
8. **Aladjev V.Z., Shirodza I.B.** *Solution of Engineering Problems in Operating Media of Language Basic for PC WANG 2200MVP.*- Harkov: HAI Press, 1988, 128 p. (in Russian with English summary)
9. **Aladjev V.Z. et. al.** *Personal Computer WANG 2200MVP.*- Kiev: USE Press, 1989, 150 p.
10. **Aladjev V.Z. et. al.** *Design of Software for PC WANG 220MVP.*- Kiev: Technika Press, 1989, 255 p.
11. **Aladjev V.Z., Shilenko V.F.** *Personal Computer IBM PC/XT.*- Kiev: USE Press, 1990, 490 p.
12. **Aladjev V.Z.** *Homogeneous Structures: Theoretical and Applied Aspects.*- Kiev: Technika Press, 1990
13. **Aladjev V.Z., Shirodza I.B.** *Personal Computers IBM PC/XT: Instrumental Tools and Software Design.*- Moscow: Higher School Press, 1991, 457 p. (in Russian with extended English summary)
14. **Aladjev V.Z.** *Scientific-Technical CADs.*- Kiev: USE Press, 1991, 432 p.
15. **Aladjev V.Z., Gershgorin N.A.** *Computational Problems on Personal Computer.*- Kiev: Technika Press, 1991, 248 p., ISBN 5-335-01064-9 (in Russian with extended English summary)
16. **Aladjev V.Z. et. al.** *Utilities for Personal Computer.*- Kiev: Technika Press, 1992, 152 p.
17. **Aladjev V.Z., Shirodza I.B.** *Computer Mixture.*- Tallinn: CAPT Press, 1992, 355 p.
18. **Aladjev V.Z., Tupalo V.G.** *Computer Reader.*- Kiev: USE Press, 1993, 448 p.
19. **Aladjev V.Z., Tupalo V.G.** *Turbo-Pascal for All.*- Kiev: Technika Press, 1993, 185 p.
20. **Aladjev V.Z., Tupalo V.G.** *Scientific Editing and Statistics on Personal Computer.*- Moscow: Mintopenergo Press, 1993, 106 p. (in Russian with extended English summary)
21. **Aladjev V.Z., Tupalo V.G.** *Computer Telecommunication.*- Moscow: Mintopenergo Press, 1993
22. **Aladjev V.Z., Tupalo V.G.** *Algebraic Computations on Computer.*- Moscow: Mintopenergo, 1993
23. *Software of IBM/360 Computers and CACS // Ed. V.Z. Aladjev.*- Tallinn: Valgus Press, 1978
24. *A Data Base Management System on the Basis of Operating System MINIOS and DBMS OKA // Ed. V.Z. Aladjev.*- Tallinn: Valgus Press, 1980, 180 p. (in Russian with extended English summary)
25. *Parallel Processing and Parallel Algorithms // Ed. V.Z. Aladjev.*- Tallinn: Valgus Press, 1981, 320 p.
26. *Parallel Processing Systems // Ed. V.Z. Aladjev.*- Tallinn: Valgus Press, 1983, 376 p.
27. *Structurally-Analytical Models and Algorithms for Recognition and Identification of Control Objects // Ed. V.Z. Aladjev.*- Kiev: Technika Press, 1993, 386 p. (in Russian with extended English summary)
28. **Aladjev V.Z., Hunt U.J., Shishakov M.L.** *Mathematics on Personal Computer // Ed. A.D. Ursul.*- Gomel: BELGUT Press, 1996, 498 p., ISBN 3-420-614023-3 (in Russian with English summary)
29. **Aladjev V.Z., Hunt U.J., Shishakov M.L.** *Fundamentals of Computer Informatics // Ed. A.D. Ursul.*- Gomel: TRG & Salcombe Esti Ltd. & Russian Academy of Noosphere, 1997, 498 p.
30. **Aladjev V.Z., Hunt U.J., Shishakov M.L.** *Basics of Informatics. Text-book. 2nd edition.*- Moscow: FILIN Press, 1999, 525 p. (in Russian with extended English summary)

31. *Efimova M.O., Hunt U.J. Geometry of Drawing: Graphical Package AutoTouch // Ed. V.Z. Aladjev.-* Gomel: VASCO & Salcombe Eesti Ltd. & TRG, 1997, 72 p. (in Russian with English summary)
32. *Aladjev V.Z., Shishakov M.L. Introduction to Media of Package Mathematica 2.2.-* Moscow: FILIN Press, 1997, 363 p., ISBN 5-89568-004-6 (in Russian with extended English summary)
33. *Aladjev V.Z., Vaganov V.A., Hunt U.J., Shishakov M.L. Introduction to Media of Mathematical Package Maple V.-* Gomel: Russian Academy of Noosphere, 1998, 470 p.
34. *Hunt U.J., Shishakov M.L. Theory of Probability and Mathematical Statistics // Ed. V.Z. Aladjev.-* Gomel: Salcombe Eesti Ltd. & TRG & Russian Academy of Noosphere, 1997, 120 p.
35. *Aladjev V.Z., Hunt U.J., Shishakov M.L. Mathematical Theory of the Classical Homogeneous Structures.-* Gomel: TRG & VASCO & Salcombe Eesti Ltd., 1998, 300 p., ISBN 9-063-56078-9
36. *Aladjev V.Z., Hunt U.J., Shishakov M.L. Questions of Mathematical Theory of the Classical Homogeneous Structures.-* Gomel: BELGUT Press, 1997, 151 p., ISBN 5-063-56078-5
37. *Aladjev V.Z., Veetyusme R.A., Hunt U.J. General Theory of Statistics.-* Tallinn: TRG & SALCOMBE Eesti Ltd., 1995, 201 p. (in Russian with extended English summary)
38. *Aladjev V.Z., Hunt U.J., Shishakov M. Course of the General Theory of Statistics // Ed. A. D. Ursul.-* Gomel: BELGUT Press, 1995, 201 p. (in Russian with extended English summary)
39. *Aladjev V.Z., Tupalo V.G. Scientific and Practical Activity of the TRG: Final Results During the 25 Years (1969 - 1993).-* Moscow: Mintopenenergo Press, 1994, 116 p. (in Russian with English summary)
40. *Aladjev V.Z., Hunt U.J., Shishakov M.L. Research Activity of the Tallinn Research Group: Scientific Report for Period 1995-1998.-* Tallinn-Gomel-Moscow: Russian Academy of Noosphere & Salcombe Eesti, 1988, 80 p. (in Russian with extended English summary)
41. *Aladjev V.Z., Vaganov V.A., Hunt U.J., Shishakov M.L. Programming in Mathematical Package Maple V.-* Tallinn-Gomel-Moscow: TRG, 1999, 470 p. (in Russian with extended English summary)
42. *Heck A. Introduction to Maple.-* Berlin-Heidelberg: Springer-Verlag, 1997, 699 p.
43. *Redfern D. The Maple Handbook.-* Berlin-Heidelberg: Springer-Verlag, 1996
44. *Burkhardt W. First Steps in Maple.-* Berlin-Heidelberg: Springer-Verlag, 1993
45. *Corless R.M. Essential Maple.-* Berlin-Heidelberg: Springer-Verlag, 1995
46. *Kamerich E. A Guide to Maple V.-* Berlin-Heidelberg: Springer-Verlag, 1995, 235 p.
47. *Rosen K. Exploring Discrete Mathematics with Maple.-* New York: McGraw-Hill, 1997
48. *Klimek G. Discovering Curves and Surfaces with Maple.-* Uppsala: Springer, 1997
49. *Zenkevich O. Finite Element Method in Engineering.-* Moscow: Mir Press, 1975
50. *Lojtsjansky L.G. Mechanics of Liquid and Gas.-* Москва: Nauka Press, 1987, 840 p.
51. *Bogdevicius M., Spruogis B. Theoretical researches of drive systems of transport machines with elastic link // Transportas, Vilnius: Technika Press, 1996, no. 2 (13), pp. 70-81.*
52. *Bezuhov N.I. Basics of the theory of elasticity, plasticity and creep.-* Moscow, 1968
53. *Zube R., Glebus S., Bogdevicius M. Theoretical Analysis of the Cothode Clearing Parameters of the Electrode Wire for Aluminium Alloys // Mechanical Engineering, Vilnius: Technika Press, 1994, no. 1, pp. 122-126.*
54. *Bogdevicius M. Berechnung instationarer Stromungen in elastisch-plastischen und visko-plastischen Rohrleitungen // Universitat Stuttgart, 1991, 48 p. (in Germany with English summary)*
55. *Bogdevicius M. Non-stationary current of liquid in elastic and ductile-plastic pipelines // Applied mechanics, Kaunas, 1993, pp. 117-124 (in Lithuanian with English summary)*
56. *Bogdevicius M. Simulation of hydraulic system of pumps by method of the characteristics // Transportas, Vilnius: Technika Press, 1997, no. 2 (15).*
57. *Char B. et al. Using Maple.-* Philadelphia: TELOS, Springer-Verlag, 1998
58. *Cornil J., Testud P. Maple V Release 4.-* Versailles: Springer-Verlag, 1997
59. *Dumas P., Courdon X. Maple.-* Le Chesnat: Springer-Verlag, 1997, 460 p.

60. **Gander W., Hrebicek J.** *Solving Problems in Scientific Computing Using Maple and MatLab.*- Zurich: Springer-Verlag, 1997, 408 p.
61. **Heal K. et al.** *Maple V: Learning Guide.*- Waterloo: Waterloo Maple Inc., 1996
62. **Monagan M. et al.** *Maple V: Programming Guide.*- Waterloo: Waterloo Maple Inc., 1996
63. *Algebraic Methodology and Software Technology // Ed. M. Johnson.*- Sydney, 1997
64. *Recent Trends in Algebraic Development Techniques // Ed. F. Parisi-Pucicce.*- Roma: Springer, 1998
65. *Graph Drawing // Ed. G. Di Battista.*- Roma: Springer-Verlag, 1997, 448 p.
66. **Betounes D.** *Partial Differential Equations for Computational Science Analysis / With Maple and Vector Analysis.*- Hattiesburg: Springer-Verlag, 1998, 530 p.
67. **Deebe E., Gunawardena A.** *Interactive Linear Algebra with Maple V.*- Houston, 1998
68. **Emtsev B.G.** *Technical Hydromechanics.*- Moscow: Nauka Press, 1978, 462 p.
69. **Paskonov V.M., Polezhaev V.I., Chudov L.A.** *The Numerical Simulation of Processes of Heat Exchange and Mass Transfer.*- Moscow: Nauka Press, 1984, 288 p.
70. **Ershov N.F., Shahverdi G.** *Finite Element Method in Problems of Hydrodynamics and Hydroelasticity.*- Leningrad: Nauka Press, 1984, 236 p. (in Russian with extended English summary)
71. **Vol'mir A.C., Kurapov B.A., Turbainvsky A.T.** *Statics and Dynamics of Complex Structures: Applied Multilevel Research Techniques.*- Moscow: Nauka Press, 1989 (in Russian with English summary)
72. **Polezhaev V.I. et al.** *Mathematical Simulation of Convective Heat Exchange and Mass Transfer on the basis of the Navier-Stokes's equations.*- Moscow: Nauka Press, 1987
73. **Litvinov V.G.** *Flow of Nonlinear Viscous Liquid.*- Moscow: Nauka Press, 1982
74. **Isachenko V.P., Osipova V.A., Sukotel A.C.** *Heat Transfer.*- Moscow: Energy Press, 1975
75. **Seegerlind L.** *Application of Finite Element Method.*- Moscow: Nauka Press, 1979
76. **Bothe K.J., Wilson E.L.** *Numerical Methods in Finite Element Analysis.*- London, 1976
77. **Bogdevicius M.** *Finite Element Method in Analysis of Rotating Pipe Conveying Fluid // Lithuanian Journal of Computational Mechanics, vol. 33, 1994, pp. 56-65.*
78. *Scientific Computing // Books-Journals-Electronic Media.*- N.Y: Springer, 1997/1998
79. **Gander W.** *Solving Problems in Scientific Computing Using Maple V and MatLab.*- Berlin: Springer-Verlag, 1997, 408 p.
80. **Govoruhin V., Tsybulin V.** *Introduction to Maple: Mathematical Package for All.*- Moscow: Mir Press, 1997 (in Russian with extended English summary)
81. **Prohorov G. et al.** *Symbolic Calculations with Maple.*- Moscow: Petit Press, 1997
82. **Flegontov A., Zajtsev V.** *Discretely-Group Methods of Integration of the Ordinary Differential Equations.*- Leningrad: Automation Institute, 1991 (in Russian with extended English summary)
83. **Abramowitz M., Stegun I.** *Handbook of Mathematical Functions.*- New York: Dover, 1965
84. **Spanier J., Oldham K.** *An Atlas of Functions.*- Washington: Hemisphere, 1987.
85. **Bathe K.J., Wilson E.L.** *Numerical Methods in Finite Element Analysis, New York, 1976.*
86. **Malinin N.N.** *The Applied Theory of Plasticity and Creep.*- Moscow, 1975, 399 p.
87. **Morozov E.M., Nikishkov G.P.** *Finite Element Method in Mechanics of Corrupting.*- Moscow, 1980
88. **Marlet R., Marcal P.** *Finite element analysis of nonlinear structures / J. Struct. Div. Proc. ASCE, 94, 1968, pp. 2081-2105*
89. **Waszczyszyn Z., Cichon Cz., Radwanska M.** *Metoda elementow skonchonych w statecznosc konstrukcji.*- Warszawa, 1990 (in Polish with English summary)
90. **Nicolaidis R.** *Maple V: A Comprehensive Introduction.*- Cambridge Univ. Press, 1996
91. **Judson P.** *Calculus Explorations Usung Maple.*- Saunders College Publishing, 1996
92. **Zwillinger D.** *Standard Maple Interactive.*- CRC Press, 1998
93. **Zachary J.** *Introduction to Scientific Programming: Computational Problem Solving Using Maple and C.*- TELOS, 1996

94. **Tocci C., Adams S.** *Applied Maple for Engineers and Scientists.*- Artech House, 1996
95. **Robertson J.** *Engineering Mathematics with Maple.*- N.Y.: McGraw-Hill, 1996
96. **Parker R.** *Maple for Basic Calculus.*- Delmar Publishing, 1997
97. **Lopez R.** *Maple V: Mathematics and Its Applications.*- Berlin: Birkhäuser, 1994
98. **Karian Z.** *Probability and Statistics Explorations with Maple.*- N.Y.: Prentice Hall, 1995
99. **Etchells T.** *Mathematical Activities with Computer Algebra.*- Chartwell-Bratt, 1997
100. **Borwein J. et al.** *Interactive Math Dictionary.*- Berlin: Springer-Verlag, 1998
101. **Abell M., Braselton J.** *Maple V by Examples.*- N.Y.: Academic Press, 1994
102. **Heal K.M. et al.** *Maple V Learning Guide for Release 5.*- Berlin: Springer, 1997
103. **Monagan M. et al.** *Maple V Programming Guide for Release 5.*- Springer, 1997
104. **Anderson G.** *Applied Mathematics with Maple.*- London: Chartwell-Bratt, 1997
105. **Beitzer R.** *Engineering Analysis with Maple/Mathematica.*- Academic Press, 1995
106. **Favorin M.V.** *Moments of Inertia of Bodies.*- Moscow: Nauka Press, 1977. 509 p.
107. **Vershinsky S.E. et al.** *Dynamics of Cars.*- Moscow, 1978, 352 p.
108. **Gerts E.V.** *Dynamics of Pneumatic Systems.*- Moscow, 1985, 256 p.
109. **Bogdevicius M.** *Dynamic Characteristics of Vibration-absorbing Elements with Permanent Magnets // Second Intern. Conf. of Mech. Engineering.*- Vilnius, 1997, **2**, pp. 5-28.
110. **Basharin A.V., Postnikov Ju.V.** *Examples of Calculation of Automatized Drive on Computers.*- Leningrad, 1990, 512 p. (in Russian with extended English summary)
111. **Aladjev V.Z., Bogdevicius M.A.** *Application of the package Maple V for solution of physical and technical problems // Internat. Conf. TRANSBALTICA-99, April 1999, VGTU, Vilnius*
112. **Aladjev V.Z., Hunt U.J.** *A Workstation for mathematician // Int. Conf. TRANSBALTICA-99, April 1999, VGTU, Vilnius, pp. 392-395. (in Russian with extended English summary)*
113. **Aladjev V.Z., Hunt U.J.** *A Workstation for mathematician // Internat. Conf. "Perfecting of mechanisms of management", April 1999, ISZ, Grodno (in Russian with extended English summary)*
114. **Aladjev V.Z., Shishakov M.L.** *Programming in package Maple V // 2nd Intern. Conf. "Computer Algebra in Fundamental and Applied Researches and Education".- Minsk: BSU, 1999, pp. 10-12.*
115. **Aladjev V.Z., Shishakov M.L.** *Automated working place of the mathematician // 2nd Intern. Conf. "Computer Algebra in Fundamental and Applied Researches and Education".- Minsk: BSU, 1999*
116. **Aladjev V.Z., Shishakov M.L., Trohova T.A.** *Educational computer laboratory of the engineer // Proc. of the 8-th Byelorussian mathematical conference, vol. 3.- Minsk: BSU, 2000, pp. 154-162.*
117. **Aladjev V.Z., Shishakov M.L., Trohova T.A.** *Applied aspects of the theory of homogeneous structures // Proc. of the 8-th Byelorussian mathematical conference, vol. 4.- Minsk: BSU, 2000*
118. **Aladjev V.Z., Shishakov M.L., Trohova T.A.** *Modelling in the classical homogeneous structures // Proc. of Internat. Conf. on Mathematical Modelling (MKMM-2000).- Herson, Ukraine, 2000*
119. **Aladjev V.Z., Shishakov M.L., Trohova T.A.** *Modeling in mathematical package Maple V // Proc. of Internat. Conf. on Mathematical Modelling (MKMM-2000).- Herson, Ukraine, 2000*
120. **Aladjev V.Z., Shishakov M.L., Trokhova T.A.** *Workstation for solution of systems of differential equations // 3rd Int. Conf. "Differential Equations and Applications".- St.-Petersburg, 2000*
121. **Aladjev V.Z., Shishakov M.L., Trokhova T.A.** *Computer laboratory for engineering researches // International Conference ACA-2000.- St.-Petersburg, 2000*
122. **Aladjev V.Z., Bogdevicius M.A., Hunt U.J.** *A workstation for mathematicians // Lithuanian conference TRANSPORT-2000.- Vilnius: Technika Press, 2000, pp. 274-282.*
123. **Aladjev V.Z.** *Interactive Course of General Theory of Statistics.*- Tallinn: The Intern. Academy of Noosphere, The Baltic Branch, 2000, 300 p. + 2 diskettes for Windows, ISBN 9985-60-866-6
124. **Aladjev V.Z., Bogdevicius M.A.** *Solution of Physical and Technical, and Mathematical Problems with Package Maple V.*- Vilnius: Technika Press, 1999, 686 p., ISBN 9986-05-398-6

125. **Aladjev V.Z., Bogdevicius M.A.** *Maple 6: Solution of Engineering and Physical, Statistical and Mathematical Problems.*- Moscow: Laboratory of Base Knowledge, 2001, 800 p. + CD
126. **Aladjev V.Z.** *Computer Algebra // Alpha*, no. 1 (April), 2001: The lecture given in *Institute of Modern Knowledge and the Grodno State University, October - November 2000*, Grodno, Byelorussia; lecture text is presented on CD [127].
127. **Aladjev V.Z., Shishakov M.L.** *A Workstation for Mathematicians.*- Moscow: Laboratory of Base Knowledge, 2000, 751 p. + CD, ISBN 5-93208-052-3 (in Russian with extended English summary)
128. **Bogdevicius M.A.** *Simulation of Dynamic Processes in Hydraulic, Pneumatic and Mechanical Drivers and Their Elements / Research Report no. 541.*- Vilnius: Technika, 2000
129. **Maple 6: Installation Guide.**- Waterloo: Waterloo Maple Inc., 2000, ISBN 1-894511-02-6
130. **Heal K. et al.** *Maple 6: Learning Guide.*- Waterloo: Waterloo Maple Inc., 2000
131. **Monagan M. et al.** *Maple 6: Programming Guide.*- Waterloo: Waterloo Maple Inc., 2000
132. **Tan K.S. et al.** *Symbolic C++: An Introduction to Computer Algebra Using Object-Oriented Programming.*- Berlin: Springer, 2000, 672 p., ISBN 1-85233-260-3
133. **Gerhard J. et al.** *MuPAD Tutorial: A Version and Platform Independent Introduction.*- Paderborn: Springer, 2000, 362 p., ISBN 3-540-67546-9
134. **Aladjev V.Z., Vaganov V.A., Shishakov M.L., Hunt U.J.** *A Workstation for Mathematicians.*- Gomel-Tallinn-Moscow: International Academy of Noosphere, 1999, 605 p.
135. **Ben-Israel A., Gilbert R.** *Computer-Supported Calculus*, in 2 volumes.- Berlin: Springer, 2000
136. **Oracle WebDB.** Version 2.0 Evaluation.- Redwood Shores: Oracle Corp., CD, 1999
137. *Computer Science. Newsletter*, no 1-3.- Berlin: Springer, 2000, [textbooks@springer.de](mailto:textbooks@springer.de), [http://www.springer.de/customers/textbook\\_inspect.html](http://www.springer.de/customers/textbook_inspect.html)
138. **Bettini C. et al.** *Temporal Databases with Multiple Granularities.*- Milan: Springer, 2000, 200 p.
139. **Feuerlicht G.** *Object-Relational Database Management.*- Sydney: Springer, 2000, 350 p.
140. **Kuper G., Libkin L., Paredaens J.** *Constraint Databases.*- Berlin: Springer, 2000, 427 p.
141. **Dajtbegov D.** *Software for Processing of Statistical Data.*- Moscow: Finance and Statistics, 1984
142. **Rouanet H. et al.** *New Ways in Statistical Methodology.*- Paris, 1998, 276 p.
143. **McPherson G.** *Statistics in Scientific Investigation: Its Basis, Application and Interpretation.*- Berlin: Springer-Verlag, 1990, 666 p. ISBN 3-540-97137-8
144. **Mokhtari M.** *MatLab 5.2 & 5.3 and Simulink 2&3 for Engineers.*- Paris: Springer, 2000, 730 p.
145. **Redfern D., Campbell C.** *The MatLab 5 Handbook.*- Waterloo: Sprinder, 1998, 488 p.
146. *SAS Institute Publications.*- Cary: SAS Institute Inc., 1994
147. *SAS/STAT User`s Guide.* Vol. 1-2.- Cary: SAS Institute Inc., 1990
148. *STATISTICA.*- Tulsa: StatSoft Inc., 1994
149. **Härdle W., Klinke S., Müller M.** *XploRe - Academic Edition: The Interactive Statistical Computing Environment.*- Berlin: Springer-Verlag, 2000, 526 p. (CD with *XploRe-Learning Guide*)
150. *S-Plus: Academic Edition Version.*- Cambridge: Mathsoft Inc., Springer, 2000, 560 p.
151. **Berenson M. et al.** *Intermediate Statistical Methods and Applications: Computer Package Approach.*- New Jersey: Prentice-Hall, 1983
152. **Dixon W.J. et al.** *BMDP Biomedical Computer Programs.*- Berkeley, 1981
153. **Voelki K.E., Gerber S.B.** *Using SPSS for Windows.*- Berlin: Springer-Verlag, 1999, 228 p.
154. **Brockwell P.J., Davis R.A.** *ITSM for Windows: A User`s Guide to Time Series Modelling and Forecasting.*- Berlin: Springer-Verlag, 1994
155. **Polasek W.** *EDA - Explorative Datenanalyse.*- Berlin: Springer-Verlag, 1994
156. **Karian Z.A., Dudewicz E.J.** *Modern Statistical Systems and GPSS Simulation.*- N.Y.: Springer-Verlag, 1998, 560 p., ISBN 0-8493-3927-7.
157. *The Seventh International Conference Computational Finance 2000.*- London: London Business

- School, June 2000; <http://www.lbs.ac.uk/cf2000>
158. **Tanner M.A.** *Tools for Statistical Inference.*- Berlin-Heidelberg: Springer-Verlag, 1994
  159. **Kobayashi M.** *Mathematica: An Introduction to Statistics and Probability.*- Tokyo: Toppan, 1994, 232 p., ISBN 4-8101-8932-5 (includes a diskette)
  160. *Teabevihik. No. 4.*- Tartu: Eesti Statistika Selts, 1994 (in Estonian)
  161. **CRC Standard Mathematical Tables and Formulae // Ed. D. Zwillinger.**- Berlin: Springer, 1995, 832 p., ISBN 0-8493-2479-3
  162. **Devore J.** *Probability and Statistics for Engineering and the Sciences.* 4th edition.- N.Y., 1995
  163. **McCall R.** *Fundamental Statistics for the Behavioral Sciences.* 5th ed.- N.Y.: Harcourt Brace, 1990
  164. **Malliavin P.** *Stochastic Analysis.*- Paris: Springer, 1997, 343 p., ISBN 3-540-57024-1
  165. **Moeschlin O. et al.** *Experimental Stochastics.*- Berlin-London: Springer, 1998, CD, 206 p.
  166. **Simonoff J.S.** *Smoothing Methods in Statistics.*- New York: Springer, 1998, 338 p.
  167. **Whittle P.** *Probability via Expectation.*- Cambridge: Springer, 2000, 352 p.
  168. **COMPSTAT 2000 - Proceedings in Computational Statistics // Eds. J. Bethlehem et al.**- The Netherlands: Springer, 2000, 540 p., ISBN 3-7908-1326-5
  169. **Kil'dishev G.S.** *General Theory of Statistics.*- Moscow: Statistics Press, 1980
  170. **McPherson G.** *Statistics in Scientific Investigation: Its Basis, Application and Interpretation.*- Berlin-Heidelberg: Springer-Verlag, 1990, 666 p., ISBN 3-540-97137-8
  171. **Bertoin J. et al.** *Lectures on Probability Theory and Statistics.*- Paris: Springer-Verlag, 1999
  172. **Mittelhammer R.** *Mathematical Statistics for Economics and Business.*- N.Y.: Springer-Verlag, 1999
  173. **Simonoff J.S.** *Smoothing Methods in Statistics.*- New York: Springer, 1998, 338 p.
  174. **Aladjev V.Z.** *Interactive Encyclopaedia of Cellular Automata.*- Tallinn: The International Academy of Noosphere, The Baltic Branch, 2006 (in preparation).
  175. **Lakin G.F.** *Biometrics.*- Moscow: Higher School Press, 1990, 352 p.
  176. **Kudrjavtsev V.A., Demidovich B.P.** *Brief Course of Higher Mathematics.*- M.: Nauka Press, 1989
  177. **Fattahi A.** *Maple V Calculus Labs.*- London: Brooks/Cole, 1998, 276 p.
  178. **MAPLETECH: Special Issue on Industrial Mathematics and Industrial Applications of Maple.**- Allentown: Air Products and Chemicals, Inc., 1998
  179. **Clegg F.** *Simple Statistics: A Course Book for the Social Sciences.*- Cambridge: Camb. University Press, 1990, 200 p., ISBN 0-521-28802-9
  180. **Coombes K. et al.** *Differential Equations with Maple.*- New York: John Wiley & Sons, 1996
  181. **Ellis W. et al.** *Maple V Flight Manual: Tutorial for Calculus, Linear Algebra and Differential Equations.*- London: Brooks/Cole Publishing Co., 1996, 376 p.
  182. **Barrow D. et al.** *Solving ODE with Maple.*- London: Brooks/Cole Pub Co., 1997
  183. **Greene R.** *Classical Mechanics with Maple.*- New York - Heidelberg: Springer-Verlag, 1995
  184. **Adams S., Tocii C.** *Maple Talk.*- New York: Prentice-Hall, 1996, 346 p.
  185. **Chiang K., Mann H.** *Maple V for Physicists.*- Berlin: Springer-Verlag, 1996, 256 p.
  186. **Cohen J., Hagin F.** *Calculus Explorations with Maple V.*- New York: Prentice-Hall, 1995
  187. *Computational statistics // Eds. W. Härdle et al.,* 2002, ISSN 0943-4062
  188. *Journal of Classification,* ISSN 0176-4268, ISSN 1432-1343 (electronic)
  189. **Capinski M., Zastawniak T.** *Probability Through Problems.*- London: Springer, 2001, 300 p.
  190. **Emery M. et al.** *Lectures on Probability Theory and Statistics.*- Paris: Springer, 2000, 349 p.
  191. **Halloran M., Berry D.** *Statistical Models in Epidemiology, the Environment and Clinical Trials.*- Berlin: Springer, 2000, 274 p., ISBN 0-387-98224-2
  192. **Jacod J., Protter P.** *Probability Essentials.*- Paris: Springer, 2000, 200 p., ISBN 0-387-98649-9
  193. **Steele M.J.** *Stochastic Calculus and Financial Applications.*- Philadelphia: Springer, 2000, 335 p.
-

194. *Journal of Nonparametric Statistics*, ISSN 1048-5252
195. *Journal of Statistical Computation and Simulation*, ISSN 0094-9655
196. *Statistics: An Journal of Theoretical and Applied Statistics*, ISSN 0233-1888
197. *Stochastics and Stochastics Reports*, ISSN 1045-1129
198. **Pecquet L.** *A First Course in Magma - the Computer Algebra System.*- Paris-Berlin: Springer, 2001
199. *Computer Algebra in Scientific Computing - CASC-2000 / Eds. V.G. Ganzha et al.*- Munchen: Springer, 2001, 439 p., ISBN 3-540-41040-6
200. **Gatermann K.** *Computer Algebra Methods for Equivariant Dynamical Systems.*- Berlin: Springer, 2000, 153 p., ISBN 3-540-67161-7
201. **Grabmeier J. et al.** *Handbook of Computer Algebra: Foundations, Application, Systems.*- Heidelberg: Springer, 2001, 400 p., ISBN 3-540-65466-6
202. **Langtangen H.P. et al.** *Advances in Software Tools for Scientific Computing.*- Oslo-Berlin: Springer, 2000, 360 p., ISBN 3-540-66557-9
203. **Scott B.** *Maple for Environmental Sciences.*- Melburn-Berlin: Springer, 2001, 360 p.
204. **Jain S.K. et al.** *Basic Linear Algebra with MatLab.*- Pittsburgh-Heidelberg: Springer, 2001, 305 p.
205. **MathCAD 2000 Professional: The Worldwide Standard for Technical Design: Student Version.**- Cambridge: Mathsoft Inc., 1999, 337 p. + CD, ISBN 3-540-14859-0
206. *Encyclopedia of Biostatistics // Eds. P. Armitage, T. Colton*, in 6 vol.- Oxford-Boston-New-York: Wiley, 1998, 4898 p., ISBN 0471-97576-1
207. **Aladjev V.Z.** *Computer algebra // Alpha*, no. 1 (April), 2001
208. **Aladjev V.Z., Bogdevicius M.A.** *Maple 6: Solution of Mathematical, Statistical and Engineering-Physical Problems.*- Tallinn-Vilnius: Vilnius Technical University, 2001, 942 p. + CD
209. *Informatiology problems of mankind in XXI century.*- Moscow: Informatiology Press, 2000
210. **Aladjev V.Z.** *Modern computer algebra for simulation of transport systems // Proc. Int. Conf. TRANSBALTICA-2001.*- Vilnius: Technika Press, 2001
211. **Abell M.** *Differential Equations with Maple V.*- London: Academic Press, 1999
212. **Articolo G.** *Partial Differential Equations and Boundary Value Problems with Maple V.*- N.Y.: AP Professional, 1998, ISBN 012-064475-4
213. **Davenport J et al.** *Computer Algebra.*- N.Y.: Academic Press, 1993
214. **Hawley W.** *Foundations of Statistics.*- N.Y.: Saunders College Publishing, 1996
215. **Kirk R.** *Statistics: An Introduction.*- N.Y.: Saunders College Publishing, 1999
216. **Ross. S.** *Introduction to Probability and Statistics for Engineers and Scientists.*- N.Y.: Academic Press, 1999, ISBN 012-598472-3
217. **Roussas G.** *A Course in Mathematical Statistics.*- N.Y.: Academic Press, 1997
218. **Freund R., Wilson W.** *Regression Analysis.*- N.Y.: Academic Press, 1998
219. **Hays W.** *Statistics.*- N.Y.: Saunders College Publishing, 1993
220. **Newmark J.** *Statistics and Probability in Modern Life.*- N.Y.: Saunders College Publishing, 1998
221. **Betounes D.** *Differential Equations: Theory and Applications with Maple.*- N.Y.: Springer, 2001
222. **Cornill J.-M., Testud P.** *An Introduction to Maple V.*- Paris: Springer, 2001, 470 p.
223. **Sampath S.** *Sampling Theory and Methods.*- Alpha Science International Ltd., 2001, 184 p.
224. **Stahel W.A.** *Statistische Datenanalyse.*- Zurich: ETH Zentrum, 2000, 375 p., ISBN 3-528-26653-8
225. **Bunday B.** *Basic optimization methods.* School of mathematical Sciences, Univ. of Brodfard, 1984
226. **Murauskas G., Shioshys V.** *Nonlinear programming.* Vilnius university, Vilnius, 1991, 131 p.
227. **Luenberger D.G.** *Linear and nonlinear programming.* 2nd. Addison-Wesly, Massachusetts, 1984
228. **Polak E.** *Computational Methods in Optimization: A Unified Approach*, Academic Press, N.Y., 1971
229. **Walsh G.R.** *Methods of Optimization.* John Wiley & Sons, 1975

230. **Goldberg D.E.** *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wisley Publishing Company, INC. Reading, 1989, 412 p.
  231. **Michalewicz Z.** *Genetics Algorithms+Data Structures=Evolution Programs*. Warsaw, 1996, 432 p.
  232. **Redfern M., Betounes D.** *Mathematical Computing: An introduction to Programming Using Maple.*- Berlin: Springer, 2002, 400 p. + CD, ISBN 0-387-95331-0
  233. **Grabmeier J. et al.** *Handbook of Computer-Algebra: Foundations, Applications, Systems.*- Berlin: Springer, 2002, 400 p. + CD, ISBN 3-540-65466-6
  234. **Aladjev V.Z., Bogdevicius M.A.** *Special Questions of Operation in Environment of the Mathematical Maple Package.*- Tallinn-Vilnius: International Academy of Noosphere & Vilnius Gediminas Technical University, 2001, 215 p. + diskette, ISBN 9985-9277-2-9
  235. **Aladjev V.Z., Vaganov V.A., Grishin E.P.** *Additional functional facilities of the Mathematical Maple Package of releases 6 and 7.*- Tallinn: International Academy of Noosphere, 2002, 312 p. + diskette
  236. **Aladjev V.Z.** *Effective Operation in the Maple Package.*- Moscow: Laboratory of Base Knowledge, 2002, 350 p. + CD (in Russian with extended English summary).
  237. **Aladjev V.** *Computer Algebra System Maple: A New Software Library / ICCS*, Melbourne, 2003
  238. **Aladjev V.** *Systems of Computer Algebra: New Software ToolBox for Maple / Int. Conf. ACAT03*, Tokyo, 2003
  239. **Aladjev V.** *Computer Algebra Systems: A New Software Toolbox for Maple.*- Palo Alto: Fultus Publishing, 2004, ISBN 1-59682-000-4
  240. **Aladjev V.** *Computer Algebra Systems: A New Software Toolbox for Maple.*- Palo Alto: Fultus Publishing, 2004, ISBN 1-59682-015-2, Adobe Acrobat eBook (pdf)
  241. **Aladjev V. et al.** *Electronic Library of Books and Software for Scientists, Experts, Teachers and Students in Natural and Social Sciences.*- Palo Alto: Fultus Publishing, 2005, CD
  242. **Aladjev V.Z., Vaganov V.A.** *Systems of Computer Algebra: A New Software Toolbox for Maple.*- Tallinn: International Academy of Noosphere, 2003, 270 p., ISBN 9985-9277-6-1
  243. **Aladjev V.Z., Bogdevicius M.A., Vaganov V.A.** *Systems of Computer Algebra: A New Software Toolbox for Maple. Second edition.*- Tallinn: International Academy of Noosphere, 2004, 462 p.
  244. **Aladjev V.Z., Bogdevicius M.A.** *Computer algebra system Maple: A new software toolbox // 4th Int. Conf. TRANSBALTICA-03*, Technics Press, April 2003, Vilnius, pp. 458-466.
  245. **Aladjev V.Z.** *Computer Algebra System Maple: A New Software Library // Int. Conf. "Computer Algebra Systems and Their Applications", CASA-2003*, Saint-Petersburg, Russia, 2003.
  246. **Aladjev V., Bogdevicius M., Vaganov V.** *Systems of Computer Algebra: A New Software Toolbox for Maple // Intern. Conf. on Software Engin. Res. and Practice, SERP'04*, 2004, Las Vegas
  247. **Aladjev V.** *Computer Algebra System Maple: A New Software Library / ICCS*, Melbourne, 2003
  248. **Aladjev V.Z., Haritonov V.N.** *General Theory of Statistics.*- Palo Alto: Fultus Corporation, 2004
  249. **Maple 8 Learning Guide.**- Toronto: Waterloo Maple Inc., 2002, 308 p.
  250. **Maple 8 Introductory Programming Guide.**- Toronto: Waterloo Maple Inc., 2002, 380 p.
  251. **Maple 8 Advanced Programming Guide.**- Toronto: Waterloo Maple Inc., 2002, 382 p.
  252. **DeMarco P. et al.** *Maple Advanced Programming Guide.*- Waterloo Maple Inc., 2005
  253. **DeMarco P. et al.** *Maple Introductory Programming Guide.*- Waterloo Maple Inc., 2005
  254. **Abell M., Braselton J.** *Maple by Example*, 3rd Edition.- Berlin: Springer, 2005
  255. **Corless R.** *Symbolic Recipes: Scientific Computing with Maple.*- Waterloo Maple Inc., 2005
  256. **Adams P. et al.** *Introduction To Mathematics With Maple.*- Waterloo Maple Inc., 2004
  257. **Maple 9.5 Getting Started Guide.**- Waterloo Maple Inc., 2004
  258. **Enns R., McGuire G.** *Computer Algebra Recipes + CD with Maple Softcover.*- Berlin: Springer, 2006
  259. <http://writers.fultus.com/aladjev/source/UserLib6789.zip>
  260. <http://writers.fultus.com/aladjev/source/Archive.zip>
-

# The information about the writers

## Aladjev Victor

**Aladjev V.Z.** was born on June 14, 1942 in the town of *Grodno* (Western Byelorussia). After successful completion of the secondary school (*Grodno*) in 1959 he matriculated on the first course of physical and mathematical faculty of *Grodno State University*, and in 1962 was transferred to the Department of Mathematics of *Tartu State University* (Estonia). In 1966 he successfully finished *Tartu State University* the speciality of *Mathematician*. In 1969 he started post-graduate studies of the Estonian Academy of Science the speciality of *Probability Theory and Mathematical Statistics*, and successfully finished in 1972 two specialities "*Theoretical Cybernetics*" and "*Technical Cybernetics*". The doctoral degree of the mathematician was assigned to him for the monography "*Mathematical Theory of Homogeneous Structures and their Applications*".

From 1972 to 1990 he had a respectable positions in a number of project-technological and research organizations of *Tallinn* (Estonia). In May 1991 he established scientific company *VASCO Ltd.* and was a director; in December 1992 he became vice-president of the joint venture *Salcombe Eesti Ltd.* In 1999 he was appointed the President of the *Tallinn Research Group* (TRG), the scientific findings of which received international recognition, first of all, in the field of activities in mathematical theory of *Homogeneous Structures* (*Cellular Automata*).

**Aladjev V.Z.** is the author of more than 350 scientific and technological publications (including 65 monographies, books and collections of the articles), published in the former USSR, Estonia, Russia, Byelorussia, the Ukraine, Lithuania, GDR, Germany, Czechoslovakia, Hungary, Japan, USA and Holland. Since 1972 he is the reviewer and associate editor of the international mathematical journal "*Zentralblatt fur Mathematik*" and since 1980 a member of IAMM (*The International Association for Mathematical Modelling*, USA). He founded the *Estonian* school of mathematical theory of *Homogeneous Structures* (*Cellular Automata*) which received international recognition and covered the fundamentals of the new section of modern mathematical cybernetics.

Since October 1993 **Aladjev V.Z.** has been elected as member of Working Group *IFIP* (*International Federation for Information Processing*, USA) studying the mathematical theory of homogeneous structures and its applications. **Aladjev V.Z.** participates as a member of the organizing committee and/or a guest lecturer in many international scientific forums in mathematics and cybernetics. In April 1994 **Aladjev V.Z.** was elected as an *academician* of the *Russian Academy of Cosmonautics* in the section of *Fundamental Researches*; in September 1994 he was elected as an *academician* of the *Russian Academy of Noosphere* in the section of *Information Science*. In September 1995 **Aladjev V.Z.** was elected as an *academician* of the *Russian Academy of Natural Sciences* in the section of *Noosphere Knowledge and Technologies* and in June 1998 as an *honorary academician* of the *Russian Ecological Academy*.

In November 1997 **Aladjev V.Z.** was elected as the *academician-secretary* of the *Baltic Branch* of the *Russian Academy of Noosphere*, integrating scientists and specialists of three *Baltic countries* and *Byelorussia*, who work in the field of a complex of scientific disciplines, which are included in the doctrine about *Noosphere* (*sphere of mind*) and areas, adjacent to it, including theoretical and applied questions of the homogeneous structures problems. As a result of reorganization of the

Russian Academy of Noosphere **Aladjev V.Z.** was elected as the *First vice-president* in December 1998. In December 1999 **Aladjev V.Z.** was elected as a *foreign member* of the *Russian Academy of Natural Sciences* (RANS) in the section of *Information Science and Cybernetics*.

Most considerable scientific results of **Aladjev V.Z.** are related to the mathematical theory of homogeneous structures (*Cellular Automata*) and its applications. The sphere of his scientific interests includes mathematics, information science, cybernetics, computing sciences, mathematical and theoretical biologies, physics, astronautics, cosmology, problems of universe and a number of other naturally-scientific areas of modern knowledge.

## Bogdevicius Marijonas

**Bogdevicius M.A.** was born on January 2, 1958 in the city of *Vilnius (Lithuania)*. After successful completion of secondary school (*Vilnius*) in 1976 he entered the Mechanical Technological Faculty of *Vilnius Civil Engineering Institute* (now *Vilnius Gediminas Technical University*), which he finished with honours in 1981 in the speciality of "*Building and road machines and equipment*". During his studies **M. Bogdevicius** actively participated in research work in the field of computational mathematics and received the diploma of the first degree in the Republican conference of student research in 1980.

**M. Bogdevicius** started working for the *Vilnius Branch of Experimental Scientific Institute of Mechanical Engineering*. Then he started working for the faculty of Mechanical Technological of *Vilnius Civil Engineering Institute*. From 1983 to 1988 he was an assistant of the chair, 1988-1991 - a senior teacher, 1991 - 1995 - a associate professor. Since 1995 **M. Bogdevicius** is the Head of the Department of Transport Technological Equipment of Transport Engineering faculty of *Vilnius Gediminas Technical University*. In 1988 Bogdevicius defended a thesis on *Building and road machines at Moscow Automobile and Road Institute*. In 1990-1991 he worked on probation in *Stuttgart University (Germany)* in the Institute of Hydraulic Machines.

In 1993 in the order of nostrification the scientific degree of the doctor of engineering science was awarded to him. **M. Bogdevicius** has published more than 180 scientific and technical works; he is the author of over 25 inventions. A lot of his research is on dynamic processes in mechanical, pneumatic and hydraulic systems.

Being the Head of the Department of *Technological Transport Equipment* of *Vilnius Gediminas Technical University*, **Bogdevicius M.A.** reads the lectures on basic courses: "*Finite element method in the mechanics of continuous medium*", "*Simulation of technological transport equipment*", "*Optimization of transport machines*", "*Vehicles' dynamics*" etc. He defended thesis on "*Simulation of dynamic processes in hydraulic, pneumatic and mechanical drives and their elements*" and was conferred a degree of the habilitated doctor in engineering science in *Vilnius Gediminas Technical University* in August 2000.

In September 1998 **M. Bogdevicius** was elected as a full member of the *Tallinn Research Group (Estonia)* in the section of informatics. In December 1998 **M. Bogdevicius** was elected as a full member of the *Russian Academy of Noosphere* in the section of informatics and informational technologies, and in December 1999 - as a full member of the *International Academy of Noosphere* in the section of informational technologies. In April 2004 **M. Bogdevicius** was elected as a full member of the *International Informatization Academy* .

**M. Bogdevicius's** interests include such topics as: computation methods, methods of optimization, dynamics of mechanical, hydraulic and pneumatic systems, problem of safety of road motion as well as software for the solution of different technical and manufacturing problems. **M. Bogdevicius** pays a lot of attention to pedagogical activities as the Head of the Transport Technological Equipment at *Vilnius Gediminas Technical University*.