Lecture notes on Numerical Continuum Mechanics

Numerical Continuum Mechanics

Volume 1

Ivan V. Kazachkov and Vitaly A. Kalion

Printed by: Div. of Heat and Power Technology
Department of Energy Technology
The Royal Institute of Technology,
Stockholm, Sweden

Stockholm
2002

Lecture notes on Numerical Continuum Mechanics

Numerical Continuum Mechanics

Volume 1

Ivan V. Kazachkov, Professor,
Visiting Professor at the Division
of Heat and Power Technology, EGI,
Royal Institute of Technology, Stockholm

Vitaly A. Kalion, Associate Professor,
Faculty of Mathematics and Mechanics,
Kiev T. Shevchenko National University

Stockholm - 2002

About the authors

Both authors are Ukrainian scientists who graduated from the National Taras Shevchenko Kiev University (Faculty of Mechanics and Mathematics, Specialization: Fluid Dynamics and Heat Transfer) in the 1970s: Kazachkov in 1976 and Kalion in 1977. Professor Ivan V. Kazachkov received his Candidate's degree (Ph.D.) in Physics and Mathematics from the Kiev Taras Shevchenko National University in 1981, and the Full Doctorship (Dr. habil.) in Mechanical Engineering from the Institute of Physics of Latvian Academy of Sciences, Riga in 1991. Associate Professor Vitaly A. Kalion has a Candidate's degree (Ph.D.) in Physics and Mathematics (from the National Taras Shevchenko Kiev University, 1984).

From the start of their research careers through to the present day, the authors have experienced the different phases of the development of computers and the numerical simulation of continuum mechanics: from the exhausting scrupulous elaboration of numerical schemes and computer algorithms, through to the application of modern commercial computer codes and powerful computers. Between them, they have about 25 years of experience in the numerical simulation of continuum mechanics, including research and lecturing at several universities in Ukraine, mainly at the National Kiev T. Shevchenko University (Faculty of Mechanics and Mathematics) and at the Kiev Land Forces Institute (Faculty of Cybernetics).

They have also worked at several of the research institutes of the Ukrainian Academy of Sciences (the Institutes of: Cybernetics, Hydromechanics, Engineering Thermophysics, Electrodynamics), conducting research into turbulent flows, multiphase systems, parametrically-controlled waves on the boundary interfaces of continua, bio-fluid dynamics and the rheology of blood, etc. In addition, the accumulation of lecture materials over the years has enabled them to prepare this a short, but detailed, course of

lectures on numerical simulation in continuum mechanics for students and engineers who are interested in acquiring a basic knowledge about the formulation of problems in this field.

The authors have come to understand from their teaching experiences and many conversations at international conferences that, over the years, numerical simulation in continuum mechanics has changed dramatically, both for the better and for the worse. They were surprised to find that nowadays many researchers use computer codes in their calculations in mechanical engineering and other fields without being fully aware of the details behind the models implemented in those codes. The authors believe that this is not right way to use computer codes; a deep knowledge about the mathematical models, the boundary conditions applied in different cases, and answers to other questions concerning the development of mathematical models and the solution of the boundary value problems are important also. Furthermore, it is absolutely necessary to have some basic knowledge not only of numerical methods and computer codes, but also of the formulation of boundary-value problems, and of their mathematical classification and the methods to be applied for their solution in each particular case.

Initially, the set of lectures and the textbook were prepared in Ukrainian. In 1998, Ivan Kazachkov was given the opportunity to work as Visiting Professor at the Energy Technology Department of the Royal Institute of Technology (Stockholm), where he began to prepare this textbook in English for the students of KTH. The idea to prepare such a lecture course arose from Ivan Kazachkov's work within the Computerised Education Project at the Division of Heat and Power Technology under the supervision of Professor Torsten H. Fransson, Prefect of the Energy Technology Department and Chair of Heat and Power Technology Division.

The authors wish to thank Professor Fransson for his support and valuable assistance during the writing of this book. His careful reading of the manuscript, discussion of the

material and comments helped the authors to improve the textbook, to the extent that some chapters were completely rewritten or reorganised. In addition, it should be noted that some of the book's chapters were included in the Computerised Educational Program for the students of HPT, Department of Energy Technology, KTH.

The authors wish to thank Professor Waclaw Gudowski, Director of the Swedish Nuclear Center for the financial support of the undergraduate course in Numerical Methods in Energy Technology (Spring 2002). And the authors are grateful to Dr. M. Vynnycky for the editing of the book.

Preface

This textbook contains short but thorough review of the basic course on numerical continuum mechanics that is read in the last two years of study in mechanical engineering faculties at most universities. Conscious of the extensive literature on computational thermofluid analysis, we have tried to present in this textbook a self-contained treatment of both theoretical and practical aspects of boundary-value problems in numerical continuum mechanics.

It is assumed that the reader has some basic knowledge of university-level mathematics and fluid dynamics, although this is not mandatory. To understand the material, the reader should know the basics of differential and integral calculus, differential equations and algebra. For the advanced reader this book can serve as a training manual. Nevertheless, it may be used also as guide for practical development, investigation or simply the use of any particular numerical method, algorithm or computer code.

In each part of the book, some typical problems and exercises on practical applications and for practical work on a computer are proposed to the reader, and it is intended that these be done simultaneously. Execution of the tasks and exercises will contribute to a better understanding of the learning material and to a development of a student's practical skills. Understanding means being able to apply knowledge in practice, and it is often better to prove a result to oneself rather than to read about how to prove it (although both methods of learning are of course useful).

The first volume of the textbook contains a description of the mathematical behaviour of partial differential equations (PDEs) with their classification, which is then followed by the basics of discretization for each type of equation: elliptic, parabolic and hyperbolic.

The most effective and useful finite-difference schemes known from the literature and from the research work of the authors are outlined for each type of PDE.

Some of the most important classical problems from the field of continuum mechanics are considered, so as to show the main features of the PDEs and their physical meaning in fluid dynamics problems.

The aim of the first volume of the textbook is to teach the basics of PDEs. The book aims to provide the finite-difference approximations of the basic classes of PDE for examples from continuum mechanics, as well as to teach numerical continuum mechanics, through a deep understanding of the behaviour of each class of PDE. This, in turn, helps to clarify the features of physical processes.

The first volume of the textbook is geared more towards a forty-hour course at undergraduate level. The second volume is directed towards an approximately eighty-hour course at graduate level and requires a more advanced knowledge of mathematics and fluid dynamics.

The book is intended as an introduction to the subject of fluid mechanics, with the aim of teaching the methods required to solve both simple and complex flow problems. The material presented offers a mixture of fluid mechanics and mathematical methods, designed to appeal to mathematicians, physicists and engineers. The main objectives are:

- to introduce the theory of partial differential equations;
- to give a working knowledge of the basic models in fluid mechanics;
- to apply the methods taught to fluid flow problems.

The textbook has been prepared based on lectures given at the National T. Shevchenko Kiev University at the Mechanical-Mathematical Faculty and at several other Ukrainian

universities, in co-operation with the Division of Heat and Power Technology, Department of Energy Technology of the Royal Institute of Technology, Stockholm. Some parts of the book are included in the Computerised Education Project in Heat and Power Technology at KTH[1], in the chapter on Computational Fluid Dynamics.

The material of the first volume was taught to the graduate students of the Faculty of Mathematics and Mechanics at the National Taras Shevchenko Kiev University as a compulsory course. The course was given after detailed compulsory courses in Calculus, Mathematical Physics, Linear Algebra, Tensor Analysis and Basics of Fluid Dynamics, when the students were already aware of the basic features of PDEs and their methods of solution. For completeness, the authors have decided to include this material in their textbook also.

The second volume is based on lectures and laboratory practicals for graduate students and on special courses for undergraduate students. The material in some parts of the second volume is difficult for undergraduate students, and is directed mainly towards specialists in numerical continuum mechanics who already have some practical experience. Nevertheless, the authors have tried to present the advanced topics in as simple a way as possible and hope that this will prove successful.

The authors hope that their book will be of interest and use for readers at different levels and within other fields of interest in numerical continuum mechanics, as well as in other areas of numerical simulation. Any comments from readers will be warmly appreciated.

Stockholm and Kiev, March 2002

I.V. Kazachkov and V.A. Kalion

[1] KTH: Royal Institute of Technology, Stockholm

CONTENTS of Volume 1

CONTENTS of Volume 2 (preliminary, under elaboration)

Nomenclature and abbreviations

$\underline{A} = \{a_{jk}\}$	Matrix with elements a_{jk}
$\underline{A}^{-1}$	Matrix inverse to the matrix $\underline{A}$
$\underline{A}^{T}$	Transpose of matrix $\underline{A}$ (the rows are replaced by the columns)
$\{\underline{A}^{k}\}$	Matrix sequence
$\lvert\underline{A}\rvert$	Determinant of the matrix $\underline{A}$
$\sum_{i=1}^{i=N} a_{ik}$	The sum of all elements a_{ik} for i from $i=1$ to $i=N$
$\sum_{i \neq j} a_{ij}$	The sum of all elements a_{ij} for all $i \neq j$
α	Thermal diffusivity, [m^2/s]
ADI	Alternating-direction-implicit (scheme)
D	Numerical domain D
$\overline{D}$	Numerical domain D including boundary Γ
D_h	Discrete numerical domain (set of grid points)
D_h^0	Internal node set (set of internal grid points)
Da	Darcy number
det	Determinant
$\frac{d}{dx}, \frac{d}{dy}$	Ordinary derivatives with respect to x and y, respectively
Δx, Δy	Finite-difference steps in the coordinates x and y, respectively
$\Delta x \to 0$	Δx tends to zero
$\underline{E}$	Identity matrix (ones on the diagonal and zeros elsewhere)
e	Exponential

$\vec{F} = \{f_i\}$	Vector with elements f_i
$f(x, y)$, $u(x, y)$	Functions of two variables x, y
$f\Big\|_a^b$	$f(b) - f(a)$
$\int_\alpha^\beta f(x)dx$	Integral of function $f(x)$ in the interval from α to β
Fo	Fourier number
FDA	Finite-difference approximation
FDE	Finite-difference equation
FI	Fully implicit (scheme)
FTCS	Forward in time, central in space
FDS	Finite-difference scheme
$\varphi(M)$	The value of the function φ at the point M of the numerical domain
Gr	Grashof number
G3LS	Generalized three-layer scheme
Γ	Boundary of the numerical domain
Γ_h	Boundary node set for the numerical grid
$\underline{IA}$	Storage indicator (nonzero elements with their location in matrix $\underline{A}$)
$J_{ij}^{(k)}$	Jacobian (determinant of the transformation)
i, j	Indices (integers: 0, 1, 2, 3, …)
$(i-1)$, i, $(i+1)$	Three consecutive points of the grid in the *i*-direction
$(\ )_{ij}$	The value of a function taken at the point (*i*,*j*) on the grid
LAEA	Linear algebraic equation array
$L(u)$	Linear differential operator for the function *u*
$\underline{LV}$	Scalar product of the matrices $\underline{L}$ and $\underline{V}$
λ_k	Eigenvalues of the matrix

$\underline{\Lambda}$	Diagonal matrix with eigenvalues λ_i (Jordan canonical form)
M	Mach number
MDDE	Multidimensional diffusion equation
MFS	Method of fractional steps
μ	Dynamic viscosity factor [Pa s]
$\vec{n}$	Normal vector
Nu	Nusselt number
$N \times N$	Matrix dimension (N rows and N columns)
NX, NY	The number of the nodes in the numerical grid in the X- and Y-directions
ODE	Ordinary differential equation
O , $O(\Delta x)^2$	The order of a quantity, the order of $(\Delta x)^2$, respectively
Pe	Péclet number
p	Pressure, [Pa]
PDE	Partial differential equation
Q_v	Heat source, [W/m^3]
R , φ	The axes of a polar coordinate system
R	Universal gas constant, [m^3 Pa/mol K]
Ra	Rayleigh number
Re	Reynolds number
$\mathbf{R}_{cell}$	Grid Reynolds number
ρ	Density, [kg/m^3]
sin, cos	Sine and cosine, respectively
sh, ch	Hyperbolic sine and cosine, respectively
σ	Stress tensor, [N/m]
T	Temperature, [K]
t	Time, [sec]

$t(\underline{A})$	Trace of matrix $\underline{A}$ (the sum of the diagonal elements)
T.E.	Truncation error (for the estimation of accuracy)
$[u]_\Gamma$	The value of a function u at the boundary Γ
u_x, u_y	First-order partial derivatives of a function u with respect to x and y, respectively
u_{xx}, u_{yy}, u_{xy}	Second-order partial derivatives of a function u with respect to x and y
u_x^y, $u_{x+\Delta x}^y$	The values of a function u at the points (x, y) and $(x+\Delta x, y)$
v_{jk}^m	Function v at the point (j,k) of the numerical grid, at time step m
$\underline{V}\ \vec{v}$	Matrix $\underline{V}$ multiplied on the right by vector $\vec{v}$
(x, y)	Point with coordinates x and y in the plane (in a 2-D domain)
X, Y	The axes of a Cartesian coordinate system
(x_i, y_j)	Point with coordinates x_i and y_j in a discrete numerical domain D_h
x_0, y_0	Coordinates of the origin $O \in D$ of the grid
$x \to i$, $y \to j$	Replacement of x by i and y by j, respectively
$\angle(\vec{n}, Ox)$	The angle between the vector $\vec{n}$ and coordinate axis Ox
$\lvert\ \rvert$, $\lVert\ \rVert$	Scalar and vector modulus, respectively

Introduction

Mathematics as a science was born out of practicalities: the measurement of distance between locations, construction, navigation, etc. As a result, in ancient times it was only numerical in so far as there was a need to obtain numbers for a particular case. Nowadays, the pure mathematician is more interested in the existence of a solution, the well-posedness of a problem and the development of general approaches to problems, etc., rather than in the solution of practical problems; from a practical point of view, however, it is often considerably more important (and more expensive) to obtain a numerical solution.

Numerical solutions to real problems have always interested mathematicians. The great naturalists of the past coupled research into natural phenomena with the development of mathematical models in their investigations. Analysis of these models called for the creation of analytical-numerical methods which, as a rule, were very problem-specific. The names of these methods are a testament to the scientists who developed them: Newton, Euler, Lobachevsky, Gauss, etc.

Numerical continuum mechanics is a very active research field involving scientists from a variety of disciplines: physicists, mathematicians, engineers, computer scientists, etc. It has a long history. Euler introduced the equations that bear his name in 1755. It is worth recalling that he noted straightaway the analytical difficulties associated with these models. More than two centuries later, some of these difficulties are still important issues, generating much activity in mathematics and scientific computing, e.g. singularities in three-dimensional solutions of the incompressible Euler equations, mathematical behaviour of solutions for compressible Euler equations, etc.

Despite fundamental contributions by many outstanding scientists (Riemann, von Neumann, Lax, etc.), the basic issues concerning the mathematical understanding of the equations of continuum mechanics are still far from being fully resolved. Continuum mechanics has changed dramatically since von Neumann's work of the late 1940s, so that theoretical analysis is now intimately connected with numerical experimentation and simulation. Furthermore, progress in the speed and power of modern computers has led scientists into the development of mathematical models for ever more complex physical problems.

More recently, the intense progress of computer science has far outstripped that of contemporary engineering. If, for example, developments in computer science were to be projected onto automobile construction, then cars ought now to be travelling at the speed of light and to be consuming microgrammes of fuel for every kilometre travelled. This progress is illustrated by Fig.1, taken from Fletcher [20], where the trend in computation

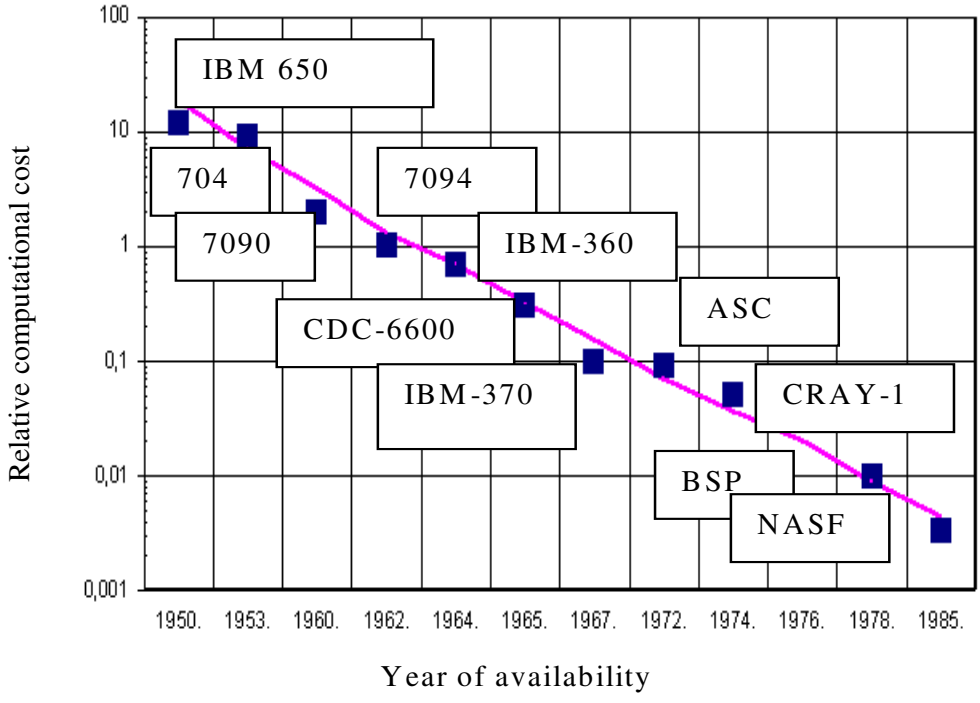


Fig. 1. The illustration of computer technics and calculation methods expansion [20].

cost for the turbulent flow around a wing profile of a viscous liquid, based on the Reynolds equations, is given by a downward sloping line. At the time that this graph made its appearance (1984), a computation on a NASF computer cost approximately $100 per minute. The cost and computation time for such calculations on an IBM-704 computer would have been about $\$10^7$ and 30 years, respectively. Computation cost diminishes approximately by a factor 10 every eight years. This tendency is being maintained at present and will probably accelerate in the future due to computer speed, which is limited only by the speed of light. Numerical methods contribute to about 40% of this speed-up. Look illustrated example :

The solution of partial differential equations is reduced to the solution of a linear algebraic equation array (LAEA), each line of which consists of 3-10 non-zero elements. Before the computer era, such systems were solved for 10-100 variables; these days, this number is 10^5-10^6 . Using old methods and modern computers, the maximum possible would be around 10^3-10^4 (for the same computation time).

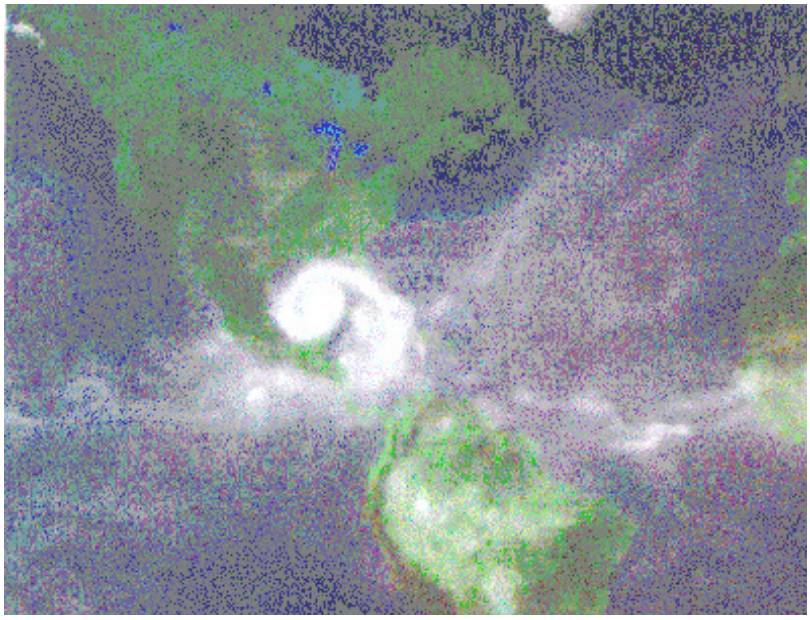


Fig. 2. Hurricane formation in the Gulf of Mexico

But despite the remarkable progress in computer science and numerical methods, some computational fluid dynamics (CFD) problems are still too complex for numerical simulation. For example, NASA has recently modified its aerospace design codes for Earth science applications, thereby speeding up supercomputer simulations of hurricane formation. An example of such a CFD simulation using a 512-processor supercomputer is shown in Fig. 2 (J. "Aviation week & space technology", August 6, 2001). Climate models and actual data from a variety of sources were merged to generate enough simulation fidelity to reproduce a hurricane forming in the Gulf of Mexico. Researchers were able to simulate the formation and movement of a hurricane based on actual data and a global atmospheric circulation model. Nevertheless, the global Earth weather forecast based on CFD atmospheric and ocean simulation is still too complex a numerical problem, and requires even more powerful computers and more precise data. In general, this is a multiphase CFD problem for a very complex geometry with dynamic boundary conditions.

Numerical methods develop naturally, their progress determined by the needs of physics and mechanics. This, in turn, has led to a reverse influence on the mathematics, and has changed it in a substantial way. In real problems, many different complications appear, most of them unconnected to the mathematics. Questions of a common nature that are important for all those working in numerical simulation are the following:

1. Of foremost importance is the correct choice of research direction. The freedom to choose is insignificant here, since the basic contours for the research direction are imposed externally. When choosing, however, it can be useful to keep in mind a simple rule-of-thumb, which divides all problems into three categories: easy, hard and very hard. Although a classification of this type may sound ridiculous, one should remember that only problems of the "hard" type are worth doing: "easy" problems will be solved in the course of your work anyway, and it is doubtful whether you will be successful in solving the "very hard" problems.

2. The second is correct formulation of mathematical model. A researcher who is able to formulate a new task correctly is more appreciated (financially also!) than one who can only solve tasks formulated by someone else. Recent graduates sometimes complain about not having had enough explanation and support from senior colleagues. The correct statement of a task is really not such a simple business, and it is often no easier than the solution. First and foremost, it is necessary to turn one's mind to the research aim because an adopted mathematical model is not something simple and unchangeable that is connected with the modelling phenomenon forever. Before writing down the differential equations, choosing the method for their solution and going to the computer, it is desirable to take a look at the problem to see whether or not it has an absurd solution. It is usual practice that a lot of calculations go to the waste basket after scrupulous analysis.

3. Success in numerical simulation requires a broad mathematical education that can only provide the ability to take on different ever-changing tasks and to solve them.

4. A perfect knowledge of mathematics, numerical methods and computers does not necessarily mean that one will always be able to solve all tasks. In many cases, some improvement of the known methods and their adaptation for the task is required. The creation of new methods becomes important here, as well as their further verification using the proven results.

5. After the completion of the computations comes the interpretation of the results obtained. When working with an inexperienced customer, it is especially important to think in advance about the accessibility of the intermediate and final results. Lots of opportunities exist for those who add a suitable interface to their work, allowing the input-output of dynamic information using graphics or, better still, multimedia. The inexperienced customer then starts to understand through mutual contact that

mathematics and the computer can give a very important component to the understanding of the problem.

6. The work should be done within a fixed period of time. A customer is often limited by a completion date and this forms a basis for decision-making. If the research is not carried out in time, then the decision will be adopted on some other basis. Furthermore, lost customer trust is often nearly impossible win back. Therefore, for the execution of a task within time constraints, it is sometimes necessary to sacrifice a more detailed model and/or a more exact, but labour-intensive, solution.

7. Team work is necessary in task execution, and it is important the results obtained can be linked seamlessly. One could mention countless examples of herd-like software development, where the distribution of tasks between parallel working executors was not sufficiently detailed in formal procedures. A simple description of final result to be obtained was not given to each executor, leading directly to delays in the execution of the whole task and thereby making it sometimes impossible to complete the task within reasonable time.

The issues mentioned above illustrate the specific character of the work in numerical continuum mechanics and lead to the conclusion that its complications outweigh those in pure mathematics insofar they require of the individual both human intellect and character, as well as a direct knowledge of mathematics.

Thus, the complexity of real physical problems and the need for advanced tools to solve them demand a systematic study of the fundamental mathematical models. In continuum mechanics, this includes not only the classical non-linear partial differential equations, such as the Euler or Navier-Stokes equations, but also modifications to them, e.g. different turbulence models, multiphase system models, flows with free boundaries, combustion, etc., which require a reduction in their numerical complexity.

The aim of this book is to improve the mathematical understanding of the models, as well as of the various known numerical methods and approaches. The main objectives are to give engineers, physicists and mathematicians a compact modern outlook on modelling in numerical continuum mechanics, on the analysis of the associated partial differential equations, on modern numerical methods and on their application to real problems.

1. Mathematical behaviour of partial differential equations (PDEs)

A research process can be illustrated as follows (see Fig.1.1):

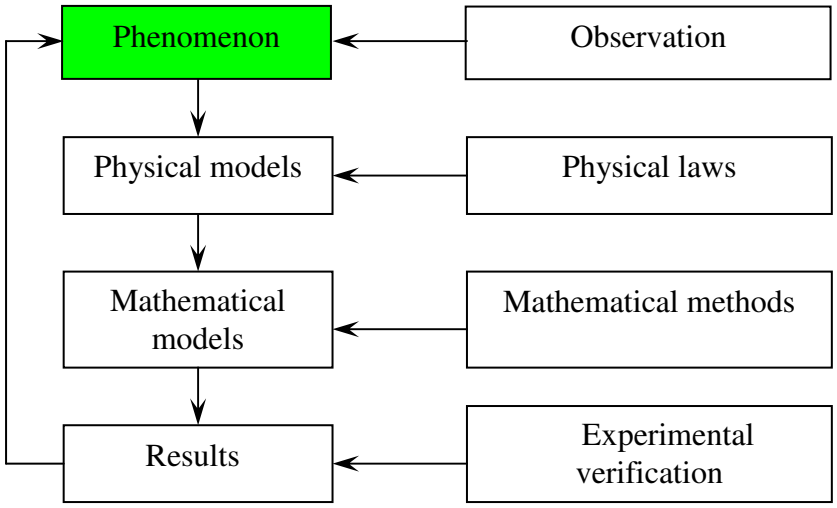


Fig.1.1. A schematic of the research process.

In general, physical problems, including those in continuum mechanics, are reduced to the solutions of partial differential equations (PDEs). Consequently, the mathematical classification of PDEs and methods for their solution constitute the basic knowledge that is required. The interconnection of mathematical methods and models to solve a problem stated is shown schematically by Fig.1.2.

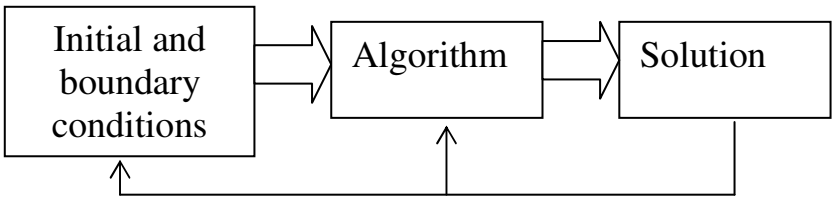


Fig.1.2. The interaction between mathematical models and methods.

In this chapter, we consider classification procedures for PDEs, and show that they may be attributed to elliptic, parabolic or hyperbolic type. Each type will be considered from the mathematical as well as the physical point of view, with a demonstration of their major descriptions and specific flows associated with each type of PDE. In addition, methods appropriate to the solution of each type will be considered.

1.1. The nature of a well-posed problem. Hadamard criteria

Before a discussion on the formal classification of PDEs, one needs to consider the task of formulation, as well as of algorithm construction, in terms of the nature of a well-posed problem. If all the Hadamard criteria:

- The solution of the problems exists;
- The solution is unique;
- The solution depends continuously on the auxiliary data (initial and boundary conditions)

are satisfied, the problem for a PDE is called well-posed.

Questions concerning the existence of a solution do not usually cause difficulties in continuum mechanics. Exceptions arise, for example, due to the absence of a solution for Laplace's equation near the centre of a source, but these can be avoided by using a transformation to move the source out of the numerical domain. Also, the analytical difficulties associated with the Euler equations are still important issues in mathematics and scientific computing, e.g. singularities in three-dimensional solutions of the incompressible Euler equations, mathematical behaviour of solutions for compressible Euler equations, etc. These and other questions will be considered in more detail later.

A non-unique PDE solution may be associated with an incomplete statement of the boundary or initial conditions (underdefinition of the problem). However, some problems

for streamlined bodies in viscous fluids should be pointed out as having non-unique solutions that are of a real and physical nature, especially in the case of transition from laminar to turbulent flow. The unique correct solution can be found under such conditions only if one knows the features of the physical process.

Remark**:** *Overdefinition of the problem (more auxiliary data than the problem requires) may cause unrealistic solutions*.

The third criterion requires that small changes in the initial or boundary conditions cause only small changes in the solution. Hadamard constructed a simple example, which shows that a solution does not always continuously depend on the initial conditions.

Example 1. Solve the Laplace equation

$$u_{xx} + u_{yy} = 0 \ , \ -\infty < x < +\infty \ , \ y \geq 0 \ . \tag{1.1}$$

for the following boundary conditions at $y=0$:

$$u(x,0) = 0, \ u_y(x,0) = (1/n)\sin(nx) \ , \ n > 0 \ . \tag{1.2}$$

Here $u_{xx} = \partial^2 u / \partial x^2$, $u_y = \partial u / \partial y$. The solution of the boundary-value problem (1.1), (1.2) is easily obtained using the analytical method of separation of variables:

$$u = (1/n^2)\sin(nx)\,\mathrm{sh}(ny) \ . \tag{1.3}$$

If the problem has been stated correctly, the solution must depend continuously on the boundary conditions. The second boundary condition of (1.2) gives that, as $n \to \infty$, the value u_y is small. As $n \to \infty$, the solution (1.3) goes behaves as $\mathbf{e}^{ny}/n^2$, growing to infinity even for small *y*, and therefore does not satisfy the first condition in (1.2). This

means that there is no continuous dependence of the solution on the boundary conditions. According to the third Hadamard criterion, the problem has been stated incorrectly. The non-fulfilment of this criterion is a strong impediment to numerical solution. This is due to the fact that the initial and boundary conditions are part of the numerical algorithm. Therefore, if the criterion is not fulfilled, the inaccuracy caused by the approximate initial and boundary conditions will propagate into the whole numerical domain of the solution. This will cause substantial deviations of the numerical solution from the real one.

1.2. Classification for linear second-order PDEs

Before proceeding with a classification of a partial differential equation, we consider a few examples of typical physical problems, after which the mathematical details will be analyzed. First, consider one-dimensional viscous non-stationary flow, which is described by the following parabolic second order equation,

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=-\frac{1}{\rho}\frac{\partial p}{\partial x}+\nu\frac{\partial^2 u}{\partial x^2},$$

where u is the flow velocity, p is the pressure, and t and x are the time and space variables, respectively. Here, $\nu=\mu/\rho$ is the kinematic coefficient of viscosity, and μ and ρ are the dynamic coefficient of viscosity and density of the liquid, respectively. The first term on the left-hand side of the equation is the local time derivative, which characterizes the velocity change in time at some fixed point of the flow region, while the second term represents convection. The convective term is non-linear and describes the velocity change in space (for coordinate x in the one-dimensional case). The second term on the right-hand side describes dissipation in the flow due to viscosity, which causes the loss of momentum in the flow. Thus, the higher the velocity, the more important is the convective term. At low velocities, the dissipation plays a more important role in the

flow. Under the same flow conditions, different flow regimes can occur due to the different physical properties: a more viscous liquid loses momentum faster than a less viscous one. It is important to note that because only the kinematic coefficient of viscosity appears in the momentum equation, liquids with different dynamic viscosities and different densities may have the same flow properties if they have the same kinematic coefficient of viscosity. For example, the densities of water and liquid metal differ by a factor of ten or more, but they can have the same kinematic coefficient of viscosity, e.g. molten steel is slightly less viscous than water.

Another example is the Laplace equation (1.1) considered above. Such an equation can be derived for a stationary two-dimensional flow when the flow velocity is small, but viscosity is large. Then, convection can be supposed to be small compared with diffusion and the convective term can be neglected, so that the elliptic second-order equation (1.1) describes the two-dimensional diffusion process. A similar equation can be obtained for the three-dimensional case. The same equation is obtained for the heat conduction process. For example, two-dimensional non-stationary heat conduction in the flow is described by the equation

$$\frac{\partial T}{\partial t}+u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}=\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right),$$

where $\{u,v\}$ is the flow velocity vector, T is the temperature, $\alpha=\dfrac{k}{c\rho}$ is the heat diffusivity coefficient, k and c are heat conductivity and specific heat, respectively. This is a mixed parabolic-hyperbolic second-order partial differential equation. But, in the stationary case, it transforms to the mixed elliptic-hyperbolic equation

$$u\frac{\partial T}{\partial x}+v\frac{\partial T}{\partial y}=\alpha\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right).$$

If the velocity is small, heat diffusivity coefficient is large and the process is stationary, all the terms to the left can be neglected and this equation simplifies to the pure elliptic Laplace equation

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0\,,$$

which describes the heat conduction process in a two-dimensional domain. The same equation is obtained for each of the velocity components for heavily viscous stationary two-dimensional flow:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0,\quad \frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}=0.$$

The same equations are obtained for a solid deformation, where $\{u,v\}$ will then be the vector of deformations in x and y, respectively.

As an example of a hyperbolic PDE, we consider the wave equation. It is easy to obtain such an equation, e.g. consider the one-dimensional oscillation of a body with mass m and let u be the coordinate of its centre. Then, $\partial u/\partial t$ is the velocity of this point and $\partial^2 u/\partial t^2$ is its acceleration. The space derivative $\partial u/\partial x$ describes the shift of the oscillating point with respect to the coordinate x and the second derivative is the gradient of this shift, which characterizes its speed. Now, assuming that an elastic force proportional to the shift gradient acts on the body, the wave equation is obtained by using Newton's second law:

$$m\frac{\partial^2 u}{\partial t^2}=k_e\frac{\partial^2 u}{\partial x^2}\,,\quad \text{or}\quad \frac{\partial^2 u}{\partial t^2}=\frac{\partial^2 u}{\partial x^2}\,,$$

where k_e is an elasticity coefficient. These are wave equations, which are second-order hyperbolic PDEs, with the second equation being written in a more simple (canonical)

form. The general solution of this equation is the simple wave $u = f(x - u_* t)$, where u_* is a constant wave velocity in x. The solution is easily verified by substitution. The main feature of the wave equation (and of all hyperbolic equations) follows from an analysis of its solution: all the perturbations spread along the lines (called characteristics) given by $x = u_* t + const$.

Many different equations have been derived for a variety of physical processes. They are described in courses on fluid dynamics, heat transfer, elasticity, etc. In addition, many different physical (and other) processes are described by similar equations. Therefore, the classification of PDEs and their methods of solution are the same for a number of diverse processes.

As was shown above, the class of a PDE may change if some of its terms are neglected, as happens when considering special conditions for the problem stated and estimating the parameters and terms in each case. For instance, the Navier-Stokes equations under different flow conditions can be in any one of the three PDE classes, or in a mixed class. Moreover, sometimes their type may change during the flow due to changes in boundary conditions, physical properties, etc., e.g. if water flows from the sea to the coast, the elliptic equations for the flow at sea are replaced by hyperbolic ones for wave flow at the coast. An air flow around a plane is described by parabolic equations in the boundary layer and by elliptic ones elsewhere if the velocity is less than the speed of sound. When the plane is moving with the speed of sound (transonic flow), the equations become parabolic everywhere. Further, hypersonic flight is described by hyperbolic PDEs.

Thus, the mathematical classification of a PDE is of paramount importance for researchers from a number of different fields in their mathematical simulation of a given problem. We use second-order partial differential equations presented in a general form for an explanation of the mathematical classification of a PDE.

Let us consider the following PDE:

$$L(u) \equiv a(x,y)u_{xx} + 2b(x,y)\ u_{xy} + c(x,y)\ u_{yy} + 2d(x,y)\ u_x + 2e(x,y)u_y + g(x,y)u = f(x,y). \tag{1.4}$$

Here a,b,c,d,e,g,f are the functions of two variables $(x,y) \in \bar{D} = D + \Gamma$. The multiplier 2 in front of b,d,e is normally introduced just for convenience because it simplifies the algebra.

For simplicity and without loss of generality, we will consider a linear equation. We note, however, that quasi-linear equations can also be considered in similar fashion; they are linear with regard to the higher order derivatives, and the coefficients a,b,c,d,e,g,f in (1.4) can also be functions of u, u_x, u_y. In the general case of a strongly non-linear PDE, classification is not so simple and sometimes may not be possible at all.

The equations of three different classes (elliptic, parabolic and hyperbolic) may be written in a general symbolic form (1.4). This classification of a second-order PDE is given by analogy with second-order curve classification in analytical geometry. The PDE class is determined, in a manner similar to a canonical cut (described in analytical geometry) for a second-order curve, by the sign of the determinant $\delta(x,y) = b^2 - a \cdot c$. Thus, only the coefficients in front of the second-order (highest) derivatives determine the class of a PDE, and any PDE can be represented in the form (1.4).

If, in the entire space $\bar{D}$, $\delta < 0$, the class is elliptic, and has the canonical form:

$$u_{\xi\xi} + u_{\eta\eta} = h_1(u_\xi, u_\eta, u, \xi, \eta). \tag{1.5}$$

If, in the entire space $\bar{D}$, $\delta = 0$, the class is parabolic, and has the canonical form

$$u_{\xi\xi} = h_2\left(u_\xi, u_\eta, u, \xi, \eta\right). \tag{1.6}$$

If, in the entire space $\bar{D}$, $\delta > 0$, the class is hyperbolic, and has two possible canonical forms. The first one (sometimes called the characteristic form) is

$$u_{\xi\eta} = h_3\left(u_\xi, u_\eta, u, \xi, \eta\right). \tag{1.7}$$

The second is

$$u_{\xi\xi} - u_{\eta\eta} = h_4\left(u_\xi, u_\eta, u, \xi, \eta\right). \tag{1.8}$$

***Remark*:** *The class of a PDE in $\bar{D}$ may change, e.g.:*

1. *The transonic flow equation*

$$\left(1 - M^2\right)\frac{\partial^2 \Phi}{\partial S^2} + \frac{\partial^2 \Phi}{\partial n^2} = 0$$

consists of three different classes of PDE depending on the local Mach number:
$M < 1$ (elliptic), $M = 1$ (parabolic), $M > 1$ (hyperbolic).

2. Stationary 2-D flow is described inside a boundary layer by a parabolic PDE:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\frac{\partial^2 u}{\partial y^2};$$

outside of the boundary layer, it is described by elliptic PDEs:

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right).$$

The different PDE categories can be associated with different kinds of physical and other problems. For example, non-stationary problems in continuum mechanics are described by parabolic or hyperbolic equations. Parabolic PDEs occur in the case of the flows with

dissipation, e.g. flows with substantial viscosity or heat conduction effects. In such cases, the solution is smooth, and for time-independent boundary conditions the gradients decrease. In the absence of dissipation, the solution of a linear PDE has constant amplitude, and for a non-linear PDE the solution even has a tendency to grow. This is a typical solution of a hyperbolic PDE. Elliptic PDEs are typical for problems describing stationary flows. Nevertheless, some stationary processes are also described by parabolic equations (boundary layer) or hyperbolic ones (inviscid supersonic flow). The following theorem is useful for practical applications:

Theorem: PDE classification remains invariable under affine, orthogonal and arbitrary generalized non-degenerate curvilinear transformations of the basic PDE (the proof is left as an exercise).

In a space $\overline{D}, N \geq 3$, consider the classification of the generalized equation

$$\sum_{j=1}^{N}\sum_{k=1}^{N} a_{jk}\frac{\partial^2 u}{\partial x_j \partial x_k} + H = 0 \ , \tag{1.9}$$

where a_{jk} are the functions of $(x_1, x_2, x_3, \ldots) \in \overline{D} = D + \Gamma$, and $H = H\left(\frac{\partial u}{\partial x_j}, u, \ldots\right)$.

By considering the matrix $\underline{A} = \{a_{jk}\}$ we arrive at the following:

- If any of the eigenvalues λ_k of the matrix $\underline{A}$ are zero, (1.9) is parabolic;
- If all $\lambda_k \neq 0$ and all have the same signs, the equation (1.9) is elliptic;
- If all $\lambda_k \neq 0$ and only one of them differs in sign, (1.9) is hyperbolic.

1.3 Classification of a first-order linear differential equation array

As was exemplified by the practical problems arising in continuum mechanics, it is rare that a problem can be reduced to just one PDE. Even when a physical or mechanical

process is described by only one higher-order PDE, it can be always replaced by a first-order equation array. Thus, to begin with, we consider the following system of linear first-order PDEs:

$$\begin{cases} a_{11}\dfrac{\partial u}{\partial x} + a_{12}\dfrac{\partial u}{\partial y} + b_{11}\dfrac{\partial v}{\partial x} + b_{12}\dfrac{\partial v}{\partial y} = f_1 \\ a_{21}\dfrac{\partial u}{\partial x} + a_{22}\dfrac{\partial u}{\partial y} + b_{21}\dfrac{\partial v}{\partial x} + b_{22}\dfrac{\partial v}{\partial y} = f_2 \end{cases} . \tag{1.10}$$

Equation (1.10) can be rewritten in vector form as

$$\underline{A}\frac{\partial \vec{w}}{\partial x} + \underline{B}\frac{\partial \vec{w}}{\partial y} = \vec{F}, \tag{1.11}$$

where $\underline{A} = \{a_{ij}\}$, $\underline{B} = \{b_{ij}\}$ are matrices, $\vec{F} = \{f_i\}$ is vector, $i, j = 1,2$ are indices. Then, introducing $|C| = \begin{vmatrix} a_{11} & b_{12} \\ a_{21} & b_{22} \end{vmatrix} + \begin{vmatrix} a_{12} & b_{11} \\ a_{22} & b_{21} \end{vmatrix}$, set $D = |C|^2 - 4|\underline{A}||\underline{B}|$, where $|\underline{A}|$ is the determinant of matrix $\underline{A}$. Following [1], let us consider the equation array (1.10) , which is:

- hyperbolic if $D > 0$,
- elliptic if $D < 0$,
- parabolic if $D = 0$.

Classification of the equation array (1.10) can also be carried out in other ways, e.g. Fletcher [20] proposed an approach based on the problem of characteristics. By this approach, the classification of an equation array depends on a solution type of the characteristic equation

$$\det\left\langle \underline{A}dy - \underline{B}dx \right\rangle = 0, \tag{1.12}$$

where $\underline{A} = \begin{Bmatrix} a_{11} & b_{11} \\ a_{21} & b_{21} \end{Bmatrix}, \underline{B} = \begin{Bmatrix} a_{12} & b_{12} \\ a_{22} & b_{22} \end{Bmatrix}$.

Equation (1.12) gives the following characteristic equation:

$$\begin{aligned} &(a_{11}b_{21} - a_{21}b_{11})\left(\frac{dy}{dx}\right)^2 - (a_{11}b_{22} - a_{21}b_{12} + a_{12}b_{21} \\ &-a_{22}b_{11})\frac{dy}{dx} + (a_{12}b_{22} - a_{22}b_{12}) = 0. \end{aligned} \tag{1.13}$$

Equation (1.13) has two solutions depending on the discriminant

$$\begin{aligned} D_1 &= (a_{11}b_{22} - a_{21}b_{12} + b_{21}a_{12} - b_{11}a_{22})^2 \; - \\ &- \; 4(a_{11}b_{21} - a_{21}b_{11})(a_{12}b_{22} - a_{22}b_{12}) \end{aligned} \; . \tag{1.14}$$

According to the classification proposed, the system (1.7) is:

- hyperbolic if $D_1 > 0$;
- parabolic if $D_1 = 0$;
- elliptic if $D_1 < 0$.

An example of such a classification may be considered for the equation array of two linear PDEs for 2-D potential compressible flow in the variables u, v :

Example 1.

$$\begin{cases} \left(\frac{u^2}{a^2} - 1\right)\frac{\partial u}{\partial x} + \left(\frac{uv}{a^2}\right)\frac{\partial u}{\partial y} + \left(\frac{uv}{a^2}\right)\frac{\partial v}{\partial x} + \left(\frac{v^2}{a} - 1\right)\frac{\partial v}{\partial y} = 0 \\ -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = 0 \end{cases}$$

The equation array has the same form as (1.10).

Substituting $a_{ij}, b_{ij} \quad i, j = 1,2$, and following Fletcher [20], we obtain

$$D_1 = 4\left(M^2 - 1\right), \quad \text{where} \quad M^2 = \frac{u^2 + v^2}{a^2},$$

from which it becomes clear that, for $M > 1$, the equation (1.14) is hyperbolic.

The construction used for the withdrawal of (1.12) can be generalized for the system of n first-order equations in two independent variables x and y. Equation (1.9) should be replaced by

$$\det\left[\underline{A}\left(\frac{dy}{dx}\right)^{(k)} - \underline{B}\right] = 0, \quad k = 3,...,n \ ; \tag{1.15}$$

where the corresponding matrices are: $\underline{A} = \left\{a_{ij}\right\}$, $\underline{B} = \left\{b_{ij}\right\}$ $i, j = 1,\ldots,n.$ The system properties depend on the solution of equation (1.15):

- If there are n real solutions, the system is hyperbolic;
- If there are ν real solutions, where $1 \le \nu \le n-1$, and no complex solutions, the system is parabolic;
- If all solutions are complex, the system is elliptic.

***Remark*:** *The latter classification applies to a second-order equation array, because such equations can be always reduced to a system of first-order equations.*

In systems with more than two variables, equation (1.10) can be also generalized. Consider the system

$$\underline{A}\frac{\partial \vec{q}}{\partial x} + \underline{B}\frac{\partial \vec{q}}{\partial y} + \underline{C}\frac{\partial \vec{q}}{\partial z} = \underline{D}, \tag{1.16}$$

where $\underline{A},\underline{B},\underline{C},\underline{D}$ are matrices, and $\vec{q}=\{u_k\}$, $k=\overline{1,n}$ are variables. The transformation of the system (1.16) yields a characteristic polynomial of order n in the form

$$\det\left[\underline{A}\lambda_x+\underline{B}\lambda_y+\underline{C}\lambda_z\right]=0, \tag{1.17}$$

where $\lambda_x,\lambda_y,\lambda_z$ determine the surface-normal vector $\vec{n}$ at a point (x,y,z). Equation (1.17) is a generalization of equation (1.12) and is an existence condition for characteristic surfaces. If the characteristic surface is a real space, then (1.17) has real solutions. If there are n such solutions, the system is hyperbolic.

A more interesting question, however, concerns the class of a PDE in a given direction. For example, putting $\lambda_x=\lambda_y=1$ and solving an equation with respect to λ_z, we find that equation (1.17) is elliptic in the z-direction, if there are complex solutions. In a similar way, the other directions may also be studied.

An example of classification for definite directions is given for the Navier-Stokes equations:

Example 2.

$$\begin{cases} u_x+v_y=0 \\ uu_x+vu_y+p_x-\dfrac{1}{\mathbf{Re}}\left(u_{xx}+u_{yy}\right)=0 \\ uv_x+vv_y+p_y-\dfrac{1}{\mathbf{Re}}\left(v_{xx}+v_{yy}\right)=0 \end{cases} \tag{1.18}$$

The equation array (1.18) is reduced to the system of the first order equations by the introduction of the auxiliary variables $R=v_x,S=v_y,T=u_y$. Thus,

$$\begin{cases} u_{\mathrm{y}} = T \\ u_{\mathrm{x}} + v_{\mathrm{y}} = 0 \\ -R_{\mathrm{y}} + S_{\mathrm{x}} = 0 \\ S_{\mathrm{y}} + T_{\mathrm{x}} = 0 \\ \dfrac{S_{\mathrm{x}}}{\mathbf{Re}} - \dfrac{T_{\mathrm{y}}}{\mathbf{Re}} + P_{\mathrm{x}} = uS - vT \\ -\dfrac{R_{\mathrm{x}}}{\mathbf{Re}} - \dfrac{S_{\mathrm{y}}}{\mathbf{Re}} + P_{\mathrm{y}} = uR - vS \end{cases} \tag{1.19}$$

The characteristic equation for (1.19) comes from the substitutions $\frac{\partial}{\partial x} \to \lambda_x$, $\frac{\partial}{\partial y} \to \lambda_y$, and the requirement that the determinant of the obtained algebraic system has to be zero:

$$\det\langle \underline{A} \rangle = \left(\frac{1}{R_c} \right) \lambda_y^2 \left(\lambda_x^2 + \lambda_y^2 \right)^2 = 0. \tag{1.20}$$

$\lambda_y = 1$ gives imaginary λ_x. If $\lambda_x = 1$ is, then λ_y is imaginary, and hence the system (1.18) is elliptic.

Further acquaintance with the general problem of PDE classification can be sought by appealing to the monographs [1, 20].

1.4 Exercises on the classification of PDEs

1. Prove the following theorem: a PDE classification remains invariable under affine, orthogonal and arbitrary generalized curvilinear non-degenerate transformations of a given PDE.

2. Prove the equivalence of the first and second canonical forms for hyperbolic equations.

3. Show that equation (1.6), written in canonical form, is parabolic.

4. Define the class of the equation: $\frac{\partial^2 u}{\partial t^2} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} = -e^{-kt}$.

5. Define the class of the equation: $\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial u}{\partial y} = 4$.

6. Define the class of the equation array for (t, x) and (t, y) for:

$$\frac{\partial u}{\partial t} + \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = 0, \qquad \frac{\partial v}{\partial t} - \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0.$$

7. Define the class of the equation: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2} = 0$

8. Define the class of the equation: $\frac{\partial^2 u}{\partial x^2} - 2\frac{\partial^2 u}{\partial x \partial y} + 5\frac{\partial^2 u}{\partial y^2} + \frac{\partial u}{\partial y} = 0$

9. Define the class of the equation: $\frac{\partial^2 u}{\partial x^2} - 6\frac{\partial^2 u}{\partial x \partial y} + 9\frac{\partial^2 u}{\partial y^2} + \frac{\partial u}{\partial x} - e^{xy} = 1$

10. Define the class of the equation: $\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2} + 7\frac{\partial u}{\partial x} - 8\frac{\partial u}{\partial y} = 0$

Optional tasks:

11. Transform the following equation to canonical form: $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2} = 0$.

12. Transform the following equation to canonical form: $\frac{\partial^2 u}{\partial x^2} - 2\frac{\partial^2 u}{\partial x \partial y} + 5\frac{\partial^2 u}{\partial y^2} + \frac{\partial u}{\partial y} = 0$.

13. Transform the following equation to canonical form:

$$\frac{\partial^2 u}{\partial x^2} - 6\frac{\partial^2 u}{\partial x \partial y} + 9\frac{\partial^2 u}{\partial y^2} + \frac{\partial u}{\partial x} - e^{xy} = 1 \ .$$

14. Transform the following equation to canonical form:

$$\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial x \partial y} + \frac{\partial^2 u}{\partial y^2} + 7\frac{\partial u}{\partial x} - 8\frac{\partial u}{\partial y} = 0 .$$

15. Replace the variables $y - \lambda_1 x = \xi + \eta$, $y - \lambda_2 x = \xi - \eta$, where λ_1, λ_2 are the solutions of a characteristic equation, and transform the function $\phi = \Phi e^{-\alpha \xi} e^{-\beta \eta}$ to reduce equation (1.4) to the canonical form (1.7) or (1.8).

2. Discretization of linear elliptic partial differential equations

As we have already seen, stationary processes are described by elliptic PDEs. The general form of an elliptic PDE is

$$L(u) = f(x, y) \ , \tag{2.1}$$

where *L* is a linear operator; or, in canonical form,

$$u_{xx} + u_{yy} = f(x, y) , \tag{2.1'}$$

where $(x, y) \in D : \bar{D} = D + \Gamma$. Here (x, y) is an arbitrary point in the numerical domain D with boundary Γ. The elliptic class requires the condition $\delta(x, y) < 0$ to be satisfied everywhere on $\bar{D}$.

The boundary conditions for PDE (2.1) or (2.1') are stated in the form (for all points $M \in \Gamma$):

- For the Dirichlet problem

$$\left[u(x, y) \right]_\Gamma = \varphi(M) . \tag{2.2a}$$

- For the Neumann problem

$$\left[\frac{\partial u}{\partial n} \right]_\Gamma = \varphi_1(M) \qquad \text{or} \qquad \left[\frac{\partial u}{\partial S} \right]_\Gamma = \varphi_2(M) \ . \tag{2.2b}$$

- For the Robin problem (mixed boundary-value problem)

$$\left[\alpha(x, y) \frac{\partial u}{\partial n} + \beta(x, y) u \right]_\Gamma = \varphi_3(M), \qquad \left[\alpha^2 + \beta^2 \right]_\Gamma > 0 . \tag{2.2c}$$

In this chapter, the main numerical procedures will be reviewed. They are used for the approximate solution of stationary problems in continuum mechanics and can be described by boundary-value problems (2.1), (2.2a); (2.1), (2.2b) or (2.1), (2.2c). A numerical scheme consists of the two steps shown in Fig.2.1:

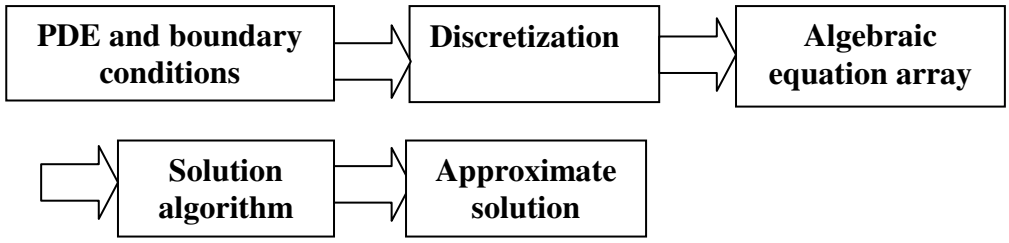


Fig.2.1. Procedure for obtaining an approximate solution.

On the first step, a PDE describing a continuous process and auxiliary (initial and boundary) conditions are transformed to a discrete system of algebraic equations; this step is called the discretization. The discretization procedure is easily identifiable for the finite-difference method (also called the grid method). However, discretization can also be carried out using the method of lines, the method of finite elements and the spectral methods, about which more will be explained later.

The next step in securing the approximate solution to a PDE is to solve the algebraic equation array obtained by discretization.

The discretization procedure also has two steps: the discretization of the domain $\bar{D}$, which means, in the case of a finite-difference method, that a corresponding discrete set of grid nodes D_h should be chosen and generated, and that the differential operators have to be replaced by difference operators. Both steps will be considered.

2.1 Grid generation

First of all, the discrete set $D_h : (x_i, y_j) \in D_h,\ u(x_i, y_j) = u_{ij}$ of grid nodes (set of points) is generated in a domain $\bar{D} = D + \Gamma$ (Fig. 2.2). The finite-difference method allows us to calculate an approximate value of the function at the internal grid nodes using their known values at the boundary grid nodes. Therefore, grid generation is one of the most important steps of the finite-difference method. We will consider the task of grid generation each time we construct an algorithm for a particular problem. Further, we will also construct algorithms for orthogonal grid generation with irregular meshes for domains of arbitrary shape. For the time being, however, we consider the simplest examples of grid generation for an arbitrary domain $\bar{D}$ with an origin at $O \in D$ in Cartesian coordinates *OXY*.

A. Rectangular grid generation

First, two sets of orthogonal lines are drawn parallel to the coordinate axes (Fig.2.2):

$$x = x_0 + i\Delta x, \quad y = y_0 + j\Delta y, \tag{2.4}$$

where (x_0, y_0) are the coordinates of the point $O \in D$; Δx, Δy are the grid steps in x and y, and $i, j = 0, \pm 1, \pm 2, \ldots$ are the indices for the points (x_i, y_j) in the numerical domain. All the nodes for both coordinates *X* and *Y* form the node net in the numerical domain $\bar{D}$. Mark the internal nodes and the nodes whose distance from the boundary Γ is no greater than $\Delta x(\Delta y)$ as $\{o\} \in D_h^0$; $\{\otimes\} \in \Gamma$, $\{*\}$, respectively. Let $\Gamma_h = \{\otimes\} \cup \{*\}$ be a discrete set of boundary grid nodes, as shown in Fig. 2.2. This is the procedure of rectangular grid generation, because all the nodes are placed on the system of two orthogonal line sets.

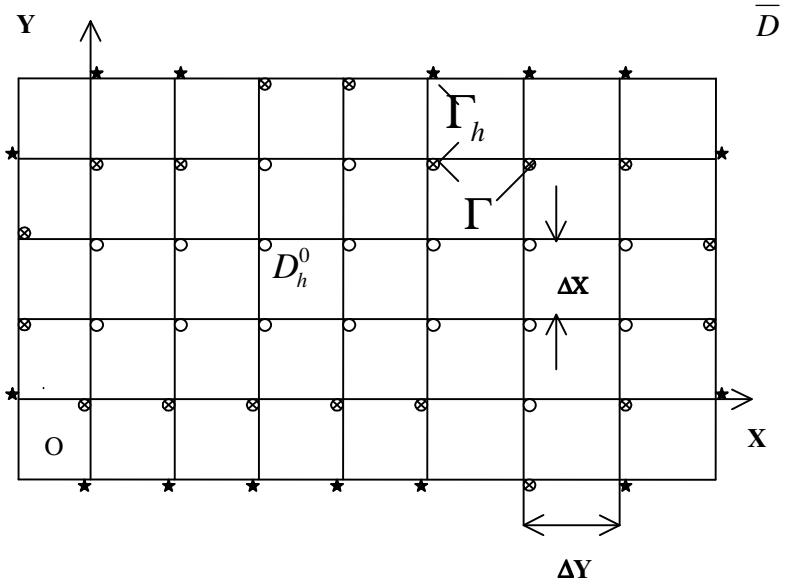


Fig. 2.2. Rectangular grid generation.

B. <u>Triangular grid generation</u>

The determination of an internal node set D_h^0 and a boundary node set Γ_h is the same as that described above. Note that, by comparison with a rectangular grid, the description of an arbitrary domain D on a triangular grid is much closer to the original one. This indicates the superiority of triangular grids when discretizing for complex curvilinear domains. The node coordinates of the triangles of the grid are given by:

$$y = \frac{\sqrt{3}}{2} j\Delta x, \;\; y = \sqrt{3}x + \frac{\sqrt{3}}{2} j\Delta x,$$
$$y = -\sqrt{3}x + \frac{\sqrt{3}}{2} j\Delta x, \;\; j = 0, \pm 1, \pm 2, \ldots; \Delta x > 0 . \tag{2.5}$$

The method of triangulation was first developed in connection with geophysical problems, where domains with very complex geometries are often considered.

Subsequently, it was also successfully applied to free boundary problems that were characterized by the appearance of local sharp deformations of the free surface. Fig. 2.3 shows a typical triangular grid.

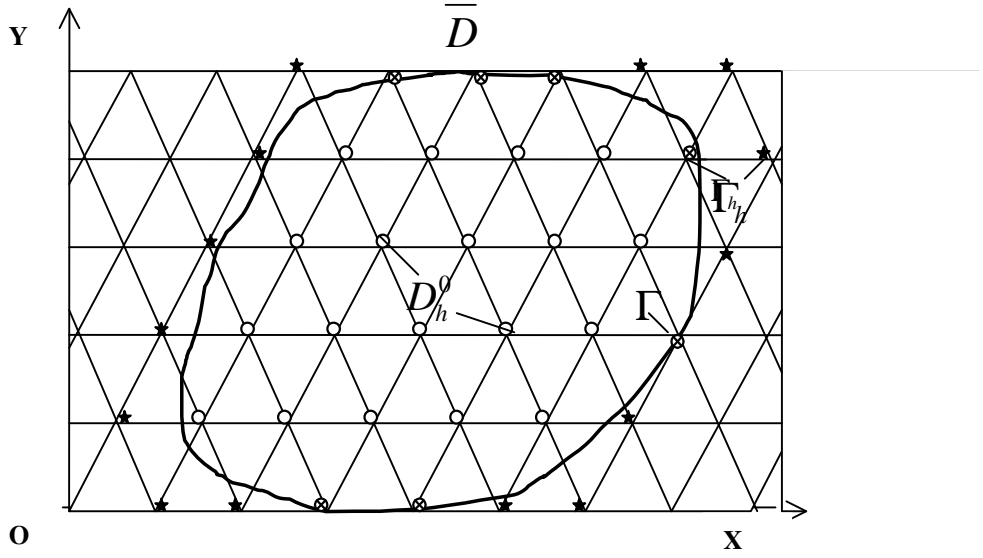


Fig. 2.3. Triangular grid generation.

C. Arbitrary grid generation

If necessary, D_h is chosen as an arbitrary set of points within $\overline{D}$, e.g. when the domain is part of a circle (see Fig. 2.4). Introduce the polar coordinates $OR\varphi$ and draw in the plane $OR\varphi$ the set of lines:

$$R = R_0 + i\Delta R,\ \varphi = \varphi_0 + j\Delta\varphi, \tag{2.6}$$

where (R_0, φ_0) are the coordinates of a point $O \in D$; ΔR is a step in the direction of OR, $\Delta\varphi$ is a step in the direction of $O\varphi$; and $i, j = 0, \pm 1, \pm 2, \ldots$ as previously.

The internal node set D_h^0 and the boundary node set Γ_h are determined in the same way as before.

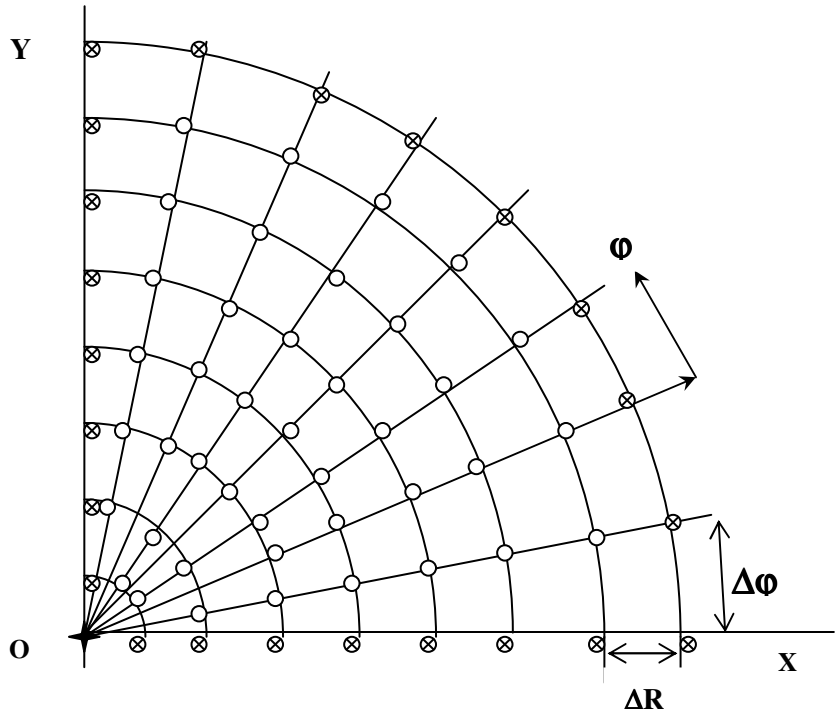


Fig. 2.4. Grid generation in polar coordinates.

2.2 Finite-difference approximations for PDEs

Finite-difference approximations for PDEs are derived on a grid by considering the equation at a point and at its surrounding neighbours. The simplest and most popular is the 5-point "criss-cross" mesh (Fig. 2.5a). In the finite-difference approach, the differential operators are represented at the internal points by difference operators that are obtained, in the simplest case, from the Taylor expansion of a function on the grid generated.

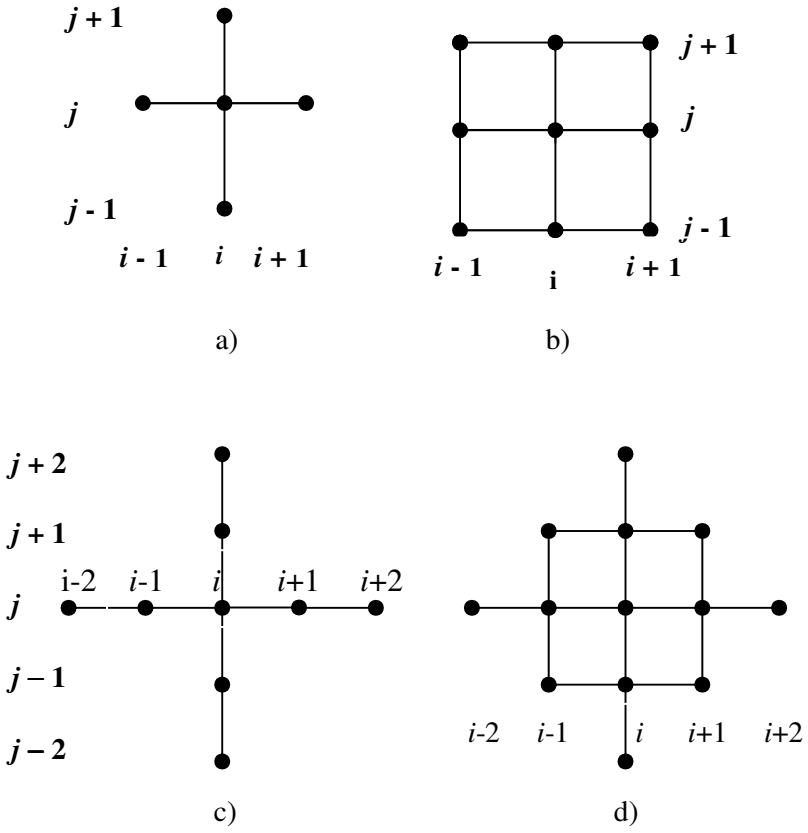


Fig. 2.5. Meshes for elliptic equations.

In a "criss-cross" mesh, the point $(x_i, y_j) \in D_h^0$ is internal if the four points surrounding it, i.e. $(x_{i-1}, y_j), (x_{i+1}, y_j), (x_i, y_{j-1}), (x_i, y_{j+1})$ all lie within $\bar{D}$. In the Ox direction, the Taylor series yields:

$$u_{i+1,j} = u_{i,j} + \frac{1}{1!}\left(\frac{\partial u}{\partial x}\right)_{ij} \Delta x + \frac{1}{2!}\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} (\Delta x)^2 + O\left((\Delta x)^3\right),$$
$$u_{i-1,j} = u_{i,j} - \frac{1}{1!}\left(\frac{\partial u}{\partial x}\right)_{ij} \Delta x + \frac{1}{2!}\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} (\Delta x)^2 + O\left((\Delta x)^3\right). \tag{2.7}$$

After simple manipulation of the equations in (2.7), the finite-difference approximation for the first order derivative with respect to *x* (in the *Ox* direction), to within an accuracy $O(\Delta x)$, is obtained as

$$\left(\frac{\partial u}{\partial x}\right)_{ij} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + O\left((\Delta x)^2\right),$$

or

$$\left(\frac{\partial u}{\partial x}\right)_{ij} = \frac{u_{i,j} - u_{i-1,j}}{\Delta x} + O\left((\Delta x)^2\right). \tag{2.8}$$

Subtracting the second expression of (2.7) from the first one gives a second-order expression with an accuracy $O\left((\Delta x)^2\right)$:

$$\left(\frac{\partial u}{\partial x}\right)_{ij} = \frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + O\left((\Delta x)^3\right). \tag{2.9}$$

By analogy, in the *y*-coordinate (in the *Oy* direction):

$$\left(\frac{\partial u}{\partial y}\right)_{ij} = \frac{u_{i,j+1} - u_{i,j}}{\Delta y} + O(\Delta y)^2, \text{ or } \left(\frac{\partial u}{\partial y}\right)_{ij} = \frac{u_{i,j} - u_{i,j-1}}{\Delta y} + O\left((\Delta y)^2\right),$$

or

$$\left(\frac{\partial u}{\partial y}\right)_{ij} = \frac{u_{i,j+1} - u_{i,j-1}}{2\Delta y} + O\left((\Delta y)^3\right). \tag{2.10}$$

The second-order derivative in the finite-difference approximation comes from the sum of the expressions in (2.7):

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O\left((\Delta x)^3\right). \tag{2.11}$$

By analogy, in the *Oy* direction:

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{ij} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O\left((\Delta y)^3\right). \tag{2.12}$$

The second-order and higher derivatives along *Ox* can be also calculated simply by the defining the derivative of a first-order (or higher) derivative:

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} = \left[\frac{\partial}{\partial x}\left(\frac{u_{i+1,j} - u_{ij}}{\Delta x}\right)\right]_{ij} = \frac{\frac{u_{i+1,j} - u_{ij}}{\Delta x} - \frac{u_{i,j} - u_{i-1,j}}{\Delta x}}{\Delta x} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O\left((\Delta x)^3\right) \tag{2.13}$$

and, in the same way, for the mixed derivatives:

$$\left(\frac{\partial^2 u}{\partial x \partial y}\right)_{ij} = \left[\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right)\right]_{ij} = \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4\Delta x \Delta y} + O\left((\Delta x)^3 + (\Delta y)^3\right). \tag{2.14}$$

Remark: *For the replacement of a mixed derivative, only internal points* (*marked* $\{o\}$) *are used. This remark requires some additional conditions for the chosen* D_h^0, *namely by changing the 5-point mesh into a 9-point mesh, similar to the "compact molecule" (Fig. 2.5b). Therefore, for further simplification, we suppose that* $b(x, y) \equiv 0$.

Now substituting (2.8)–(2.12) into (2.1) yields

$$\left[L(u)\right]_{ij} = L_h u_{ij} + O\left((\Delta x)^3 + (\Delta y)^3\right) = f_{ij}\ . \tag{2.15}$$

The finite-difference operator is then given by

$$L_h u_{ij} = A_{ij} u_{i+1,j} + B_{ij} u_{i-1,j} + D_{ij} u_{i,j+1} + E_{ij} u_{i,j-1} + F_{ij} u_{ij}, \tag{2.16}$$

where

$$A_{ij} = \frac{a_{ij}}{(\Delta x)^2} + \frac{d_{ij}}{\Delta x}, \qquad B_{ij} = \frac{a_{ij}}{(\Delta x)^2} - \frac{d_{ij}}{\Delta x}, \qquad D_{ij} = \frac{c_{ij}}{(\Delta y)^2} + \frac{e_{ij}}{\Delta y},$$

$$E_{ij} = \frac{c_{ij}}{(\Delta y)^2} - \frac{e_{ij}}{\Delta y}, \qquad F_{ij} = -\frac{2a_{ij}}{(\Delta x)^2} - \frac{2c_{ij}}{(\Delta y)^2} + g_{ij}.$$

If the mixed derivative is present in (2.1), (i.e. $b(x, y) \neq 0$), its approximation is made on the 9-point "compact molecule" mesh (Fig. 2.5b) using equation (2.14). Substituting (2.8)–(2.12) and (2.14) into (2.1) yields

$$\left[L(u)\right]_{ij} = \Lambda_h u_{ij} + O\left((\Delta x)^3 + (\Delta y)^3\right) = f_{ij}, \tag{2.17}$$

where the finite-difference operator is

$$\begin{aligned} \Lambda_h u_{ij} &= A_{ij} u_{i+1,j} + B_{ij} u_{i-1,j} + D_{ij} u_{i,j+1} + E_{ij} u_{i,j-1} \; + \\ &+ F_{ij} u_{ij} + \; G_{ij} \left[u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1} \right] . \end{aligned} \tag{2.18}$$

Compared with (2.15), this contains the additional term $G_{ij} = b_{ij} / 2\Delta x \Delta y$.

Remarks:

- *By (2.15), the accuracy of a finite-difference approximation for a differential operator depends on the accuracies of the approximations for each of the derivatives, and is equal to the order of the lowest one.*
- *The required accuracy of approximation for the differential operator predetermines the mesh type. If the required accuracy is greater than $O((\Delta x)^2, (\Delta y)^2)$, the 9-point "compact molecule" mesh may be used instead of the 5-point "cross" mesh (Fig. 2.5c), and the 13-point mesh (Fig. 2.5d) may be used instead of the 9-point mesh.*

- *D_h nodes belonging to the interior D_h^0 or the boundary Γ_h depend on the mesh chosen.*
- *The best way to determine the accuracy of an approximation for the mesh chosen is to calculate the Taylor series expansion in peripheral mesh nodes against the point of the central mesh node (i,j), with further substitution of the expressions obtained into the finite-difference approximation of the original equation[#]. For example, the Laplace equation has the following form on the 5-point mesh:*

$$u_{xx} + u_{yy} = -\frac{1}{12}\left(u_{xxxx}(\Delta x)^2 + u_{yyyy}(\Delta y)^2\right) + \ \dots\ , \tag{2.19}$$

which has accuracy $\approx O((\Delta x)^2, (\Delta y)^2)$.

Now, as an example, let us consider the finite-difference approximations for the Laplace equation and for the Poisson equation (2.1'), respectively (note that the Laplace equation may be considered as a limiting case of the Poisson equation with zero right-hand side):

$$u_{xx} + u_{yy} = 0\,, \qquad u_{xx} + u_{yy} = f(x, y)\,..$$

Using the finite-difference approximations (2.11), (2.12) for the second-order derivatives in *x* and *y* results in the following finite-difference equation for the Laplace equation:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O\left((\Delta y)^3 + (\Delta x)^3\right) = 0\,,$$

and, in a similar way, for the Poisson equation (2.1'):

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O\left((\Delta y)^3 + (\Delta x)^3\right) = f_{i,j}\,,$$

[#] This procedure is known as finding differential approximation for the finite-difference scheme (or **modified equation**), and will be considered in more detail in 5.2.2.

where $f_{i,j} = f(x_i, y_j)$. Equation (2.19) yields the following differential approximation for the finite-difference scheme (**modified equation**) for the Laplace equation,

$$u_{xx} + u_{yy} = -\frac{1}{12}\left(u_{xxxx}(\Delta x)^2 + u_{yyyy}(\Delta y)^2\right) + \ \ldots,$$

which allows to determine the accuracy of an approximation for the mesh chosen. For instance, let us consider here the simplest rectangular domain with boundaries at $x = 0,1$ and $y = 0,1$ and choose the following steps for the numerical grid: $\Delta x = \Delta y = 0.1$ and $\Delta x = \Delta y = 0.01$ (grids with 10 and 100 points in both coordinates, respectively). Then, we obtain the following modified equations for the Laplace equation on these numerical grids, respectively:

$$u_{xx} + u_{yy} = -\frac{u_{xxxx} + u_{yyyy}}{1200} + \ \ldots \quad \text{and} \quad u_{xx} + u_{yy} = -\frac{u_{xxxx} + u_{yyyy}}{120000} + \ldots$$

Corresponding modified equations can be obtained for the Poisson equation

$$u_{xx} + u_{yy} = f_{i,j} - \frac{u_{xxxx} + u_{yyyy}}{1200} + \ldots \quad \text{and} \quad u_{xx} + u_{yy} = f_{i,j} - \frac{u_{xxxx} + u_{yyyy}}{120000} + \ldots$$

The modified equations thus obtained can be used for the required grid estimation depending on the boundary conditions (whether or not large function gradients in the numerical domain may be supposed, which kind of boundary conditions is specified, etc.) and other characteristics of the boundary-value problem posed. For example, it follows directly from the modified equations mentioned above that, for the Poisson equation, one can get an idea about the required accuracy much more easily than for the Laplace equation, because it is possible to estimate the accuracy of the numerical approximation simply by comparing the deficiency term on the right-hand side with the known function

values $f_{i,j}$ at all discrete points (x_i, y_j) of the numerical domain. But the difficulty of estimating the fourth-order derivatives u_{xxxx}, u_{yyyy} still remains.

Now let us analyse some physical problems with different boundary conditions. First, the Laplace equation is considered and it is supposed that it describes the stationary heat conduction process with the Dirichlet boundary condition (2.2a) in the form:

$$x=0,\ \ u=0; \qquad x=1,\ \ u=0; \qquad y=0,\ \ u=1; \qquad y=1,\ \ u=0.$$

This means that three boundaries of the domain are kept at the same constant temperature and the fourth is held at a different constant temperature, e.g. heated wall. Based on these boundary conditions, it can be supposed that the temperature gradient in *y* is more substantial than in *x* and that u_y is of order 1:

$$u_x \approx \frac{u(0,y)-u(1,y)}{1-0} \approx 0, \quad u_y \approx \frac{u(x,0)-u(x,1)}{1-0} \approx 1.$$

Supposing that the temperature in the surroundings is zero, one can get an estimate for the highest-order derivatives, which are the same as the first-order ones if only one-sided derivatives are considered, so that $u_{xxxx} \approx 0$, $u_{yyyy} \approx 1$. Then, from the above-mentioned modified equations, the accuracy can be estimated, so as to choose the proper numerical grid. Such estimation is very rough, especially for the highest-order derivatives and can be used only for simple boundary conditions when no large gradients are expected in the numerical domain.

To continue the analysis of the heat conduction problem, we consider next the Neumann boundary condition (2.2b) in the form:

$$x=0,\ u_x=0; \quad x=1,\ u_x=0; \quad y=0,\ u_y=1; \quad y=1,\ u_y=0.$$

Now, at the boundary $y = 0$, suppose that the heat flux is constant. If the tangential derivative is prescribed at the boundary, corresponding to the other Neumann boundary condition (2.2b), it is impossible to estimate the gradients inside the domain. For the above boundary conditions, the accuracy estimation is made in a similar way to before, although now it is more precise because the first derivatives are given.

For the mixed boundary-value (Robin) problem (2.2c), it is more difficult to estimate the accuracy in this way because only algebraic relations between functions and their gradients, rather than functions or gradients, are given at the boundary, e.g.:

$$x = 0, \quad a_1 u_y + b_1 u = c_1; \quad x = 1, \quad a_2 u_y + b_2 u = c_2;$$
$$y = 0, \quad a_3 u_x + b_3 u = c_3; \quad y = 1, \quad u_x + b_4 u = c_4.$$

This is the most complex situation if all the constants a_i and b_j are non-zero. No parameter estimation is possible for the boundary conditions. In such cases, the numerical solution allows us to make the estimation of parameters only after initial trial calculations.

2.2.1 Derivation of finite-difference equations using polynomial approximations

An alternative method of finite-difference approximation for a PDE is based on the use of an approximate analytical function containing free parameters, which is built first on the mesh chosen and is then differentiated analytically. The ideal form for an approximate function is determined by an approximate analytical solution of the problem, although it is mostly polynomials that are used as such approximate functions. We will demonstrate this method for an example of a parabolic approximation by considering, as before, an equation of type (2.1),

$$L(u) = f(x, y), \tag{2.20}$$

with, as boundary conditions, one of the three possible types (2.2a)-(2.2c). The numerical domain nodes D_h are taken to be arbitrary. Some of them are located in the interior, whilst the others are at the boundary. Let us write the function f for the consecutive points $i-1,\ i,\ i+1$ in x and build the parabolic approximation of the function in the form

$$f(x) = a + bx + cx^2, \tag{2.21}$$

supposing the initial co-ordinate ($x = 0$) to be at the point i. Then, equation (2.21) written at the points $i-1,\ i,\ i+1$, respectively, yields

$$f_{i-1} = a - b\Delta x + c(\Delta x)^2,\ f_i = a\ ,\ f_{i+1} = a + b\Delta x + c(\Delta x)^2. \tag{2.22}$$

Adding the third expression to the first gives

$$c = \frac{f_{i+1} - 2f_i + f_{i-1}}{2(\Delta x)^2}, \tag{2.23}$$

and, solving for b, we obtain

$$b = \frac{f_{i+1} - f_{i-1}}{2\Delta x}. \tag{2.24}$$

At the point *i*, the first and second derivatives of (2.21) are:

$$\left(\frac{\partial f}{\partial x}\right)_i = \left[b + 2cx\right]_{x=0} = b, \tag{2.25}$$

$$\left(\frac{\partial^2 f}{\partial x^2}\right)_i = 2c. \tag{2.26}$$

Expressions (2.25) and (2.26), in tandem with (2.23) and (2.24), give exactly the same expressions as (2.9) and (2.11), which were obtained using the Taylor series expansion. If we consider f as a first-order polynomial, we get expressions for the first derivative (2.24). The difference approximations for the higher orders are obtained by using higher-order polynomials.

Remark: *The inaccuracy of a polynomial approximation is calculated by constructing a modified equation.*

2.2.2 Difference approximation by an integral approach

For this approach, the finite-difference equations (FDE) are constructed by integrating the basic equations on a given numerical grid (Fig. 2.6).

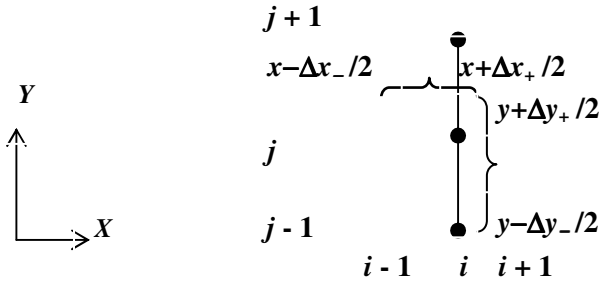


Fig. 2.6. The integration domain for the integral method.

The superiority of the integral method over a Taylor series expansion or polynomial approximation is evident in its use for non-uniform grids and curvilinear co-ordinate systems.

Control-volume methods and their modifications (e.g. particles in cell (PIC) and fluid in cell (FLIC) methods developed at Los Alamos [1]) are the ones closest to the integral method.

To consider the integral method, let us integrate the Laplace equation (1.1)

$$L(u) = u_{xx} + u_{yy} = 0$$

in the domain $D_h = [x + \Delta x_+ / 2, x - \Delta x_- / 2] * [y + \Delta y_+ / 2, y - \Delta y_- / 2]$. Because integration with respect to x and y is commutative, we integrate so that one of the double integrals can be calculated exactly:

$$L(u) = \int_{x-\Delta x_-/2}^{x+\Delta x_+/2} \left[\int_{y-\Delta y_-/2}^{y+\Delta y_+/2} \frac{\partial^2 u}{\partial y^2} dy \right] dx + \int_{y-\Delta y_-/2}^{y+\Delta y_+/2} \left[\int_{x-\Delta x_-/2}^{x+\Delta x_+/2} \frac{\partial^2 u}{\partial x^2} dx \right] dy =$$

$$= \int_{x-\Delta x_-/2}^{x+\Delta x_+/2} \left[\left. \frac{\partial u}{\partial y} \right|_x^{y+\Delta y_+/2} - \left. \frac{\partial u}{\partial y} \right|_x^{y-\Delta y_-/2} \right] dx + \int_{y-\Delta y_-/2}^{y+\Delta y_+/2} \left[\left. \frac{\partial u}{\partial x} \right|_{x+\Delta x_+/2}^{y} - \left. \frac{\partial u}{\partial x} \right|_{x-\Delta x_-/2}^{y} \right] dy \ , \tag{2.27}$$

with all further integrals being calculated numerically. Here, the limits of integration are substituted after the first integration, and all the terms are written as functions of two arguments at the corresponding points. For example, the first term on the right-hand side of equation (2.27) means the following:

$$\left. \frac{\partial u}{\partial y} \right|_x^{y+\Delta y_+/2} = \frac{\partial u}{\partial y}(x, y + \Delta y_+ / 2). \tag{2.28}$$

Thus, the points written as integral limits in (2.28) should be understood just as coordinates. The derivative $\partial u/\partial x$ can be calculated by the equation

$$u^{y}_{x+\Delta x} = u^{y}_{x} + \int\limits_{x}^{x+\Delta x} \left[\frac{\partial u}{\partial x}\right]^{y}_{x} dx\,. \tag{2.29}$$

By the mean value theorem:

$$\int\limits_{\xi}^{\xi+\Delta\xi} f(z)\ \ dz \approx f(\overline{z})\ \ \Delta\xi\,, \tag{2.30}$$

where $\overline{z} \in [\xi,\ \xi+\Delta\xi]$. The convergence of the finite-difference equation (FDE) to the partial differential equation (PDE) is guaranteed as $\Delta x, \Delta y \to 0$ (limiting guarantee of approximation). Computing the integral in (2.29) approximately using the rectangle formula, the point $x+\Delta x/2$ yields

$$u^{y}_{x+\Delta x} = u^{y}_{x} + \left[\frac{\partial u(x+\Delta x/2, y)}{\partial x}\right]\Delta x, \tag{2.31}$$

or

$$\left[\frac{\partial u}{\partial x}\right]^{y}_{x+\Delta x/2} = \left[\frac{\partial u(x+\Delta x/2, y)}{\partial x}\right] = \frac{u^{y}_{x+\Delta x} - u^{y}_{x}}{\Delta x}\,. \tag{2.32}$$

The Laplace equation (2.27) transforms to

$$\begin{aligned} L(u) = {} & \frac{2}{\Delta x_{+} + \Delta x_{-}}\left(\frac{u^{y}_{x+\Delta x_{+}} - u^{y}_{x}}{\Delta x_{+}} - \frac{u^{y}_{x} - u^{y}_{x-\Delta x_{-}}}{\Delta x_{-}}\right) + \\ & + \frac{2}{\Delta y_{+} + \Delta y_{-}}\left(\frac{u^{y+\Delta y_{+}}_{x} - u^{y}_{x}}{\Delta y_{+}} - \frac{u^{y}_{x} - u^{y-\Delta y_{-}}_{x}}{\Delta y_{-}}\right) = 0 \end{aligned} \tag{2.33}$$

With the substitutions:

$$x \to i,\ y \to j,\ x+\Delta x \to i+1,\ y+\Delta y \to j+1, x-\Delta x \to i-1,\ y-\Delta y \to j-1,$$

the finite-difference approximation (FDA) for the Laplace equation on a non-uniform grid yields the following form

$$L(u) = \frac{2}{\Delta x_+ + \Delta x_-}\left(\frac{u_{i+1,j}-u_{ij}}{\Delta x_+} - \frac{u_{ij}-u_{i-1,j}}{\Delta x_-}\right) + \\ + \frac{2}{\Delta y_+ + \Delta y_-}\left(\frac{u_{i,j+1}-u_{ij}}{\Delta y_+} - \frac{u_{ij}-u_{i,j-1}}{\Delta y_-}\right) = 0 . \tag{2.34}$$

Other methods of FDA construction, including variation, projection and variation-projection, are described in the monographs [4, 37, 45, 48, 53, 57, 59, 65, 67, 70-72].

2.3 **Difference Dirichlet boundary conditions**

Let us now examine the Dirichlet boundary-value problem for the elliptic PDE (1.4). We choose a rectangular grid D_h with D_h^0 as the set of the internal nodes and Γ_h as the set of the boundary nodes (see Fig. 2.2). The Dirichlet boundary condition (2.2a) has to be replaced with a finite-difference boundary condition (Fig. 2.7). The simplest expression is

$$u(B) = \varphi(M) , \tag{2.35}$$

where B is the boundary point nearest to the given point M . This approach is called the replacement of the boundary conditions at the nearest grid point. The accuracy is estimated here by Taylor's formula,

$$u(B)=u(M)+\frac{\delta}{1!}u_x(\zeta,y)=\varphi(M)+\frac{\delta}{1!}u_x(\zeta,y), \tag{2.36}$$

where $B=\{x+\delta,y\}$, $M=\{x,y\}$, $x<\zeta<x+\delta$, $\delta\cong\Delta x$. The inaccuracy is therefore $\approx O(\Delta x)$.

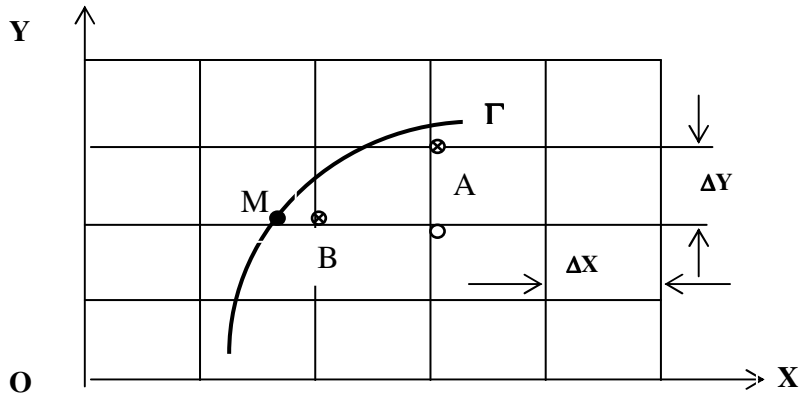


Fig. 2.7. Two-point approximation of the Dirichlet boundary condition.

To increase the accuracy of the boundary condition (2.35), one needs to use the function value $u(A)$ at the internal point $A=(x+\gamma,\Delta x)$; $\gamma=\delta+\Delta x$. The Taylor series expansion (2.36) gives, with respect to the points *A* and *B*, respectively:

$$\begin{aligned} u(B)&=\varphi(M)+\frac{\delta}{1!}u_x(M)+\frac{\delta^2}{2!}u_{xx}(M)+... \\ u(A)&=\varphi(M)+\frac{\gamma}{1!}u_x(M)+\frac{\gamma^2}{2!}u_{xx}(M)+... \end{aligned} \tag{2.37}$$

Now, the well-known Collatz formula is obtained from equations (2.37) as

$$u(B) = \frac{\Delta x \varphi(M) + \delta u(A)}{\Delta x + \delta}\ , \tag{2.38}$$

where the function value $u(A)$ is unknown and has to be calculated from formula (2.37).

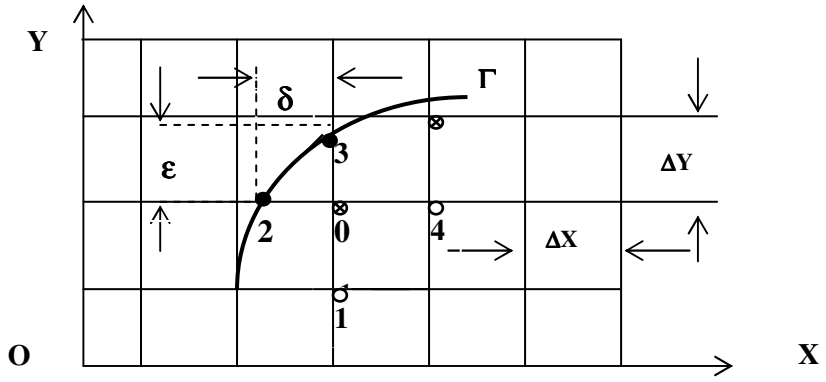


Fig. 2.8. Four-point approximation of the Dirichlet boundary condition.

The approximation of the boundary conditions at the point O is based on the points 1, 2, 3, 4 (Fig. 2.8) using the same step sizes for *Ox* and *Oy*: $\Delta x \equiv \Delta y$. The interpolation of a function of two arguments yields

$$u_0 = \frac{\varepsilon}{\varepsilon+\delta} \cdot \frac{\Delta x \varphi_2 + \delta u_4}{\Delta x + \delta} + \frac{\delta}{\varepsilon+\delta} \cdot \frac{\Delta x \varphi_3 + \varepsilon\ u_1}{\Delta x + \varepsilon} - \frac{\delta \varepsilon \Delta x f_0}{2(\varepsilon+\delta)} + O\left((\Delta x)^3\right), \tag{2.39}$$

where u_i, φ_j, f_0 are the functions u, φ, f taken at the corresponding points.

Because the accuracy of the boundary condition approximation influences the numerical solution in the entire numerical domain, one needs to approximate the boundary condition as precisely as possible. The following example of the stationary heat conduction boundary-value problem can be considered in a curvilinear domain:

Example. Consider a curvilinear numerical domain and suppose that the Dirichlet boundary condition is stated for a stationary heat conduction process given by

$$T_{xx} + T_{yy} = 0 .$$

Generating a rectangular grid, one can plot the curvilinear boundary on which constant temperature is prescribed. Thus, for constant temperature at the boundary (constant wall temperature T_w, say), the function $\varphi(M) = T_w$ is constant, and therefore $\varphi_2(M) = \varphi_3(M) = T_w$ in equation (2.39). Then, choosing the grid steps, e.g. $\Delta x = \Delta y = 0.1$, we need to calculate the boundary values at all boundary points such as **0** in Fig. 2.8. For instance, suppose that for the first boundary point $\mathbf{0_1}$: ε =0.05, δ =0.07. Then, from (2.39),

$$u_{01} = \frac{0.05}{0.05+0.07} \cdot \frac{0.1T_w + 0.07u_{41}}{0.1+0.07} + \frac{0.07}{0.05+0.07} \cdot \frac{0.1T_w + 0.05u_{11}}{0.1+0.05} + O(0.001),$$

where the double indices are used for u_{ij} and φ_{ij} to indicate that a four-point approximation of the Dirichlet boundary condition is used for every boundary point $\mathbf{0_j}$. Thus the Dirichlet boundary condition is approximated at every boundary point $\mathbf{0_j}$ of the numerical domain. The function values u_{0j} at the boundary are expressed through the boundary value $\varphi(M)$ and the functions u_{ij} taken at the two closest internal points, so that for the next point $\mathbf{0_{j+1}}$ the point 4*j* is used again, and so on going clockwise in Fig. 2.8. Therefore the whole system of the above-mentioned equations for u_{0j} allows us to exclude all internal points and to obtain a relation between just the boundary points $\varphi_k = \varphi(M_k)$, and $\varphi_{k+1} = \varphi(M_{k+1})$ if needed; whether it is or not depends on the numerical method used for the solution of the original partial differential equation inside numerical domain. In many cases it is not needed, especially for implicit schemes, as will become clear from the numerical algorithms considered later.

Using explicit numerical schemes it is useful to derive explicit boundary conditions. For this purpose, the above-considered equations can be written for two adjacent boundary points as

$$u_{0j}=\frac{\varepsilon_j}{\varepsilon_j+\delta_j}\cdot\frac{\Delta x\varphi_{2j}+\delta_j u_{4j}}{\Delta x+\delta_j}+\frac{\delta_j}{\varepsilon_j+\delta_j}\cdot\frac{\Delta x\varphi_{3j}+\varepsilon_j u_{1j}}{\Delta x+\varepsilon_j}+O\left((\Delta x)^3\right),$$

$$u_{0j+1}=\frac{\varepsilon_{j+1}}{\varepsilon_{j+1}+\delta_{j+1}}\cdot\frac{\Delta x\varphi_{2j+1}+\delta_{j+1}u_{4j+1}}{\Delta x+\delta_{j+1}}+\frac{\delta_{j+1}}{\varepsilon_{j+1}+\delta_{j+1}}\cdot\frac{\Delta x\varphi_{3j+1}+\varepsilon_{j+1}u_{4j}}{\Delta x+\varepsilon_{j+1}}+O\left((\Delta x)^3\right).$$

Now, u_{4j} can be expressed from the first equation through u_{0j} and u_{1j} and substituted into the second equation. Then, the same can be done at step (j+2) for the u_{4j+1}, and so on, until the last point of the boundary where $u_{4N}=u_{11}$.

2.4 Difference Neumann boundary conditions

The finite-difference Neumann boundary condition is considered in Fig. 2.9 below.

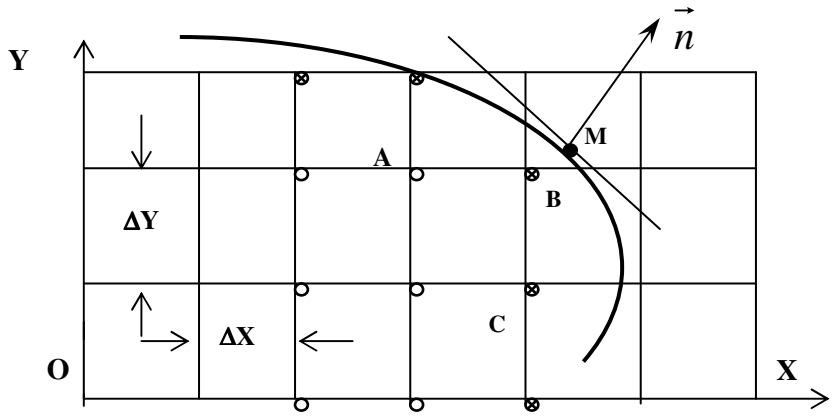


Fig. 2.9. Finite-difference Neumann boundary condition.

Let B denote a boundary node with coordinates (x_i, y_j) and suppose M is the contour point nearest to point B; let A be an interior node with coordinates $\left(x_{i-1}, y_j\right)$, and C the boundary node $\left(x_i, y_{j-1}\right)$, with $\vec{n}$ as the outward normal to the curvilinear boundary Γ at the point M.

Let $\angle\left(\vec{n}, Ox\right) = \alpha$, so that $\angle\left(\vec{n}, Oy\right) = \frac{3}{2}\pi + \alpha$. Replace the boundary condition

$$\left[\frac{\partial u}{\partial \vec{n}}\right]_\Gamma = \varphi(M) \tag{2.40}$$

with a difference condition at the boundary node B. By the definition of directional differentiation

$$\frac{\partial u}{\partial \vec{n}} = \frac{\partial u}{\partial x}\cos\alpha + \frac{\partial u}{\partial y}\cos\left(\frac{3}{2}\pi + \alpha\right) = \frac{\partial u}{\partial x}\cos\alpha + \frac{\partial u}{\partial y}\sin\alpha \ . \tag{2.41}$$

Let the direction $\vec{n}$ at the point B be the same as at the point M. The distance $BM \cong O\left(\Delta x\right)$, and therefore this assumption causes an inaccuracy no greater than $O\left(\Delta x\right)$:

$$\frac{\partial u(M)}{\partial \vec{n}} = \frac{\partial u(B)}{\partial \vec{n}} + O\left(\Delta x\right) \ . \tag{2.42}$$

Consequently:

$$\begin{aligned} &\frac{u\left(x_i, y_j\right) - u\left(x_{i-1}, y_j\right)}{\Delta x}\cos\alpha + \frac{u\left(x_i, y_j\right) - u\left(x_i, y_{j-1}\right)}{\Delta y}\sin\alpha + \\ &+O\left(\Delta x + \Delta y\right) = \varphi(M) \end{aligned} \tag{2.43}$$

or

$$\frac{u_{ij}-u_{i-1,j}}{\Delta x}\cos\alpha+\frac{u_{ij}-u_{i,j-1}}{\Delta y}\sin\alpha=\varphi(M)\ , \tag{2.44}$$

with the inaccuracy of the approximation (2.44) being $\approx O(\Delta x+\Delta y)$.

Remark: *The FDE in (2.44) has to be written for all boundary nodes* $(i,j)\in\Gamma_h$. *Therefore, for a complicated contour* Γ, *the replacement of the Neumann boundary condition with a finite-difference one is not so straightforward, although it becomes so if the function* $\varphi(M)$ *is either the same for the entire contour* Γ *or constant.*

2.5 A theorem on the solvability of difference-equation arrays

Consider a boundary-value problem in a domain $\bar{D}=D\cup\Gamma$ for the equation

$$L(u)\equiv au_{xx}+bu_{yy}+cu_x+du_y\ +\ gu=f(x,y), \tag{2.45}$$

where a,b,c,d,g are two-dimensional functions of (x,y). If $a,b>0$, equation (2.45) is elliptic. Let $g\le 0$ and consider the Dirichlet boundary condition:

$$u\big|_{\Gamma}=\varphi(M). \tag{2.46}$$

The following theorem (Krylov, et. al. [45]) on the solvability of a system of finite difference equations (2.15), (2.16) for the boundary-value problem (2.45), (2.46) is very important for practical work:

Theorem: *If the discrete domains* D_h, D_h^0 *and* Γ_h *have been constructed using the procedure mentioned above, and the grid steps* Δx *and* Δy *are small enough to satisfy the conditions* $A_{ij}, B_{ij}, \ldots E_{ij}\ge 0$ *for the system of finite-difference equations (2.16), the boundary-value problem (2.45)-(2.46) has a solution which is unique.*

The proven uniqueness of the numerical solution of the boundary-value problem stated is especially important because it ensures that the solution obtained is real. Otherwise, considerable additional work is required to prove this fact in every practical case, and sometimes this is harder than finding the solution itself. The non-existence or non-uniqueness of a solution to a boundary-value problem can be caused by peculiarities in the equations or boundary conditions (differential or difference).

2.6 Runge rule for the practical estimation of inaccuracy

We consider the grid D_h ($\frac{\Delta y}{\Delta x} = const$), and suppose that the approximate solution $u_h(x,y)$ exists and that the inaccuracy of the exact solution $u(x,y)$ with respect to Δx is known, so that

$$\varepsilon_h(x,y) = u(x,y) - u_h(x,y) = K(x,y)\cdot(\Delta x)^p + O\left((\Delta x)^{p+m}\right), \tag{2.47}$$

where $K(x,y) \neq 0$ on $\overline{D}$. For the finite-difference equation (FDE) $m = 1$ (for the finite element equation it can be $m = 3$). Introduce

$$\varepsilon_{2h}(x,y) = K(x,y)(2\Delta x)^p = 2^p \cdot \varepsilon_h(x,y) \tag{2.48}$$

and write the exact solution for the steps Δx and $2\Delta x$:

$$\begin{aligned} u(x,y) &= u_h(x,y) + \varepsilon_h(x,y) \ , \\ u(x,y) &= u_{2h}(x,y) + 2^p \varepsilon_h(x,y). \end{aligned} \tag{2.49}$$

Subtraction of the second equation in the equation array (2.49) from the first one results in the Runge rule for the practical estimation of inaccuracy:

$$\varepsilon_h(x,y) = \frac{u_h(x,y) - u_{2h}(x,y)}{2^p - 1} \ . \tag{2.50}$$

An approximate solution may then be elaborated with the Runge rule:

$$\tilde{u}_h(x,y) = u_h(x,y) + \frac{u_h(x,y) - u_{2h}(x,y)}{2^p - 1} + O\left((\Delta x)^{p+m}\right). \tag{2.51}$$

This is the Richardson formula.

Remarks:

- *In practice, the Runge rule is satisfied if*

$$\left| 2^p \frac{u_h - u_{h/2}}{u_{2h} - u_h} \right| < 0.1, \tag{2.52}$$

  this, in turn, is a confirmation of the condition $K(x,y) \neq 0$ *on* $\overline{D}$.

- *The Runge rule is also used for a practical estimation of an approximation inaccuracy for quadrature formulations and ordinary differential equations (ODE).*

Because elliptic PDEs normally describe stationary physical processes, the next chapter is devoted to the detail study of the methods for the solution of stationary boundary-value problems. The basic and most popular numerical methods will be described and analyzed.

3. Methods of solution for stationary problems

As was discussed earlier, stationary problems in continuum mechanics are often described by elliptic equations (linear or non-linear), e.g. the FDE for stationary viscous flow can be written as

$$\underline{A}(\vec{v})\vec{v} = \vec{b}\,, \tag{3.1}$$

where the vector $\vec{v}$ consists of unknown node values. The matrix $\underline{A}$ consists of algebraic coefficients which appear as a result of discretization and may depend also on the solution $\vec{v}$. The matrix $\underline{A}$ normally contains many zero elements (and is termed a sparse matrix). The vector $\vec{b}$, in a similar way, consists also of the algebraic coefficients of discretization and known values of $\vec{v}$ on the boundary Γ_h.

Remark: *As a rule, the matrix $\underline{A}$ is sparse and the non-zero elements are located near the main diagonal.*

The non-linear system (3.1) can be solved iteratively, e.g. using the Gauss-Seidel method (the method of consecutive elimination of variables). Note that more effective here may be the introduction of an external iteration with a linearisation of the system at each step. This allows the use of the direct methods, e.g. the Gauss method. Newton's method also belongs to such a class of iterative methods.

If the matrix $\underline{A}$ does not depend on $\vec{v}$, then instead of equation (3.1) there will be a linear (linearized) equation array in the form

$$\underline{A}\vec{v} = \vec{b} \tag{3.2}$$

Then, depending on the form of the matrix $\underline{A}$ (tridiagonal, alternating tridiagonal and pentadiagonal, sparse), equation (3.2) can be solved by a variety of splitting procedures

(marching Thomas methods, factorization methods, etc.), which we will consider in further detail.

The dimension of a matrix $\underline{A}$ is $N \times N$, where N is the number of internal nodes in the numerical domain. The structure of matrix $\underline{A}$ (Fig. 3.1) depends on the algorithm for constructing vector $\vec{v}$. For the typical 9-point "compact-molecule" numerical mesh presented in Fig. 2.5b, the index "*i*" numbers the nodes with respect to Ox (from 1 to NX), and index "*j*" numbers the nodes with respect to Oy (from 1 to NY). Note: the higher numbers relate to regions located near the domain boundary.

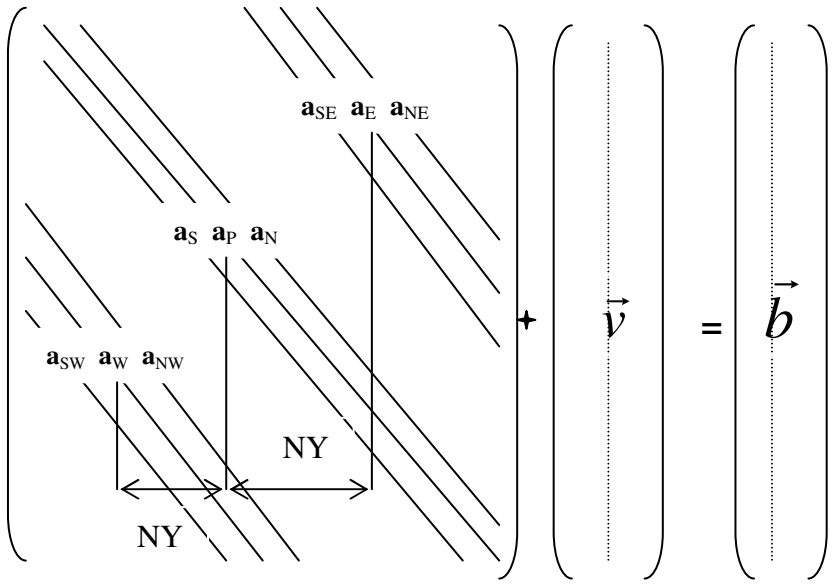


Fig. 3.1. Scheme for equation (3.1) or (3.2) on a "compact molecule" mesh.

Let us introduce the following node assignations:

$$(i, j) \to P;\ (i-1, j) \to W;\ (i+1, j) \to E;\ (i, j-1) \to S;\ (i, j+1) \to N;$$
$$(i-1, j-1) \to SW;\ (i-1, j+1) \to SE;\ (i+1, j-1) \to NW;\ (i+1, j+1) \to NE.$$

If the vector $\vec{v}$ is constructed in such a way that the nodes follow each other along a line, line by line, corresponding to an increase of the indices "*j*" and "*i*", respectively, then the

matrix $\underline{A}$ has the three central non-zero diagonals. The matrix $\underline{A}$ also has three non-zero diagonals located symmetrically on either side of the central non-zero diagonals. Thus, the number of non-zero diagonals corresponds to the number of nodes in the mesh.

Remark: *the diagonals* $\boldsymbol{a}_{SW}$, $\boldsymbol{a}_{NW}$, $\boldsymbol{a}_{SE}$, $\boldsymbol{a}_{NE}$ *are absent on a "cross" mesh.*

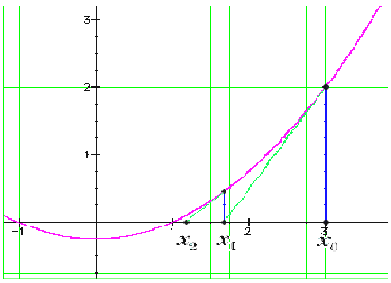
Some iterative methods (nearly all of them are applied to systems like (3.1)) are conceptually similar to the introduction of an equivalent non-stationary form of the problem stated. Since the non-stationary solution is of little importance as regards a stationary problem, it is reasonable to modify the time-dependent terms in such a way as to optimize the rate at which convergence to a stationary solution is obtained. Because the inaccuracy of the transient time-dependent iterations does not influence the inaccuracy of the stationary solution to be obtained, large time-steps (arbitrary in general) can be chosen. This is the main idea of the pseudo-nonstationary methods, which are considered later.

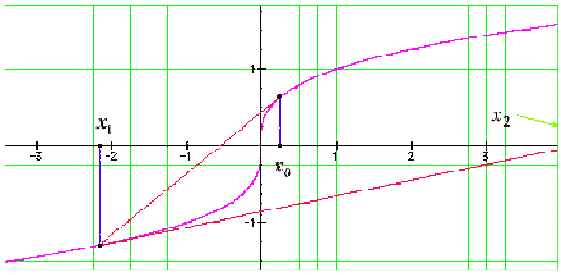
From the mathematical point of view, the introduction of an equivalent non-stationary form of the given stationary problem constitutes the replacement of an elliptic PDE by a parabolic one. Iterative methods for the solution of an equivalent non-stationary problem, which converges to the stationary one, allow us to obtain the solution of a stationary problem. In addition, this permits us to trace the characteristics of the transition from a non-stationary to a stationary process, as well as from a parabolic- to an elliptic-class PDE.

3.1 Newton's method

Newton's method for finding roots of functions was originally a method for approximating the roots of functions or, equivalently, solutions to equations of the

common form $f(x)=0$. The method is easy to understand and is very efficient at finding the solution to a given equation. Starting with a value x_0 that is suspected to be close to a root, the tangent to the curve $y=f(x)$ through the point $(x_0, f(x_0))$ can be expected to lie close to the curve in the vicinity of x_0, implying that the point where the line crosses the x-axis should be very close to the root. Since the slope of curve is the derivative $f'(x_0)$, the point of intersection is easy to compute: its x-value is $x_1 = x_0 - f(x_0)/f'(x_0)$.





x_1 should therefore be a better approximation to the actual root than x_0. However, if it is not close enough, the method is used again, this time on x_1, to obtain a better approximation x_2. The method is then used iteratively until the required accuracy is achieved. For many functions, this iterative method works very well, although not for all functions, e.g. it does not work for any function that does not have a root. Sometimes, it does not work for functions that do have roots, for example, the cube root function: searching for a root of the function $f(x)=\sqrt[3]{x}$ using any initial value x_0, it is found that the estimates will not converge. The graph above shows how, with an initial estimate x_0 near 0, the subsequent estimates x_1, x_2 and so on, approach infinity.

Even for those functions where it does work, it is still possible to choose a bad initial point x_0, e.g. such that $f'(x_0)$=0. Nevertheless, the method is on the whole very

effective, and there are various criteria that can help to predict when the method will succeed.

Now let us consider the matrix equation (3.1) in the form:

$$\vec{R} = \underline{A}(\vec{v})\vec{v} - \vec{b} = 0. \tag{3.3}$$

Then, the basic formulation of the algorithm for Newton's method

$$\vec{v}^{(k+1)} = \vec{v}^{(k)} - \left(\underline{J}^{(k)}\right)^{-1} \vec{R}^{(k)}, \tag{3.4}$$

where k is the iteration number, and $J_{ij}^{(k)} = \partial R_i^{(k)} \big/ \partial v_j^{(k)}$ is the Jacobian, which is the determinant of transformation, e.g. in a two-dimensional case:

$$J \equiv \frac{\partial(x,y)}{\partial(\xi,\eta)} \equiv \begin{vmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{vmatrix}.$$

Now, introduce the vector difference $\Delta\vec{v}^{(k+1)} = \vec{v}^{(k+1)} - \vec{v}^{(k)}$ and rewrite equation (3.4) in the following form:

$$\underline{J}^{(k)} \Delta\vec{v}^{(k+1)} = -\vec{R}^{(k)}. \tag{3.5}$$

Vector equation (3.5) is a linear equation array with respect to a correction vector $\Delta\vec{v}^{(k+1)}$ for each iteration. The solution at each iteration step is

$$v^{(k+1)} = v^{(k)} + \omega \Delta v^{(k+1)}. \tag{3.6}$$

Here $0 \le \omega \le 1$ is introduced as an underrelaxation parameter. One characteristic of Newton's method is known (e.g. [92]) to be quadratic convergence, which is expressed mathematically as

$$\left\| \vec{v}^{(k+1)} - \vec{v}_c \right\| \approx \left\| \vec{v}^{(k)} - \vec{v}_c \right\|^2 \tag{3.7}$$

Here, $\vec{v}_c$ is the exact solution of equation (3.1). A block-scheme for Newton's method is presented in Fig. 3.2.

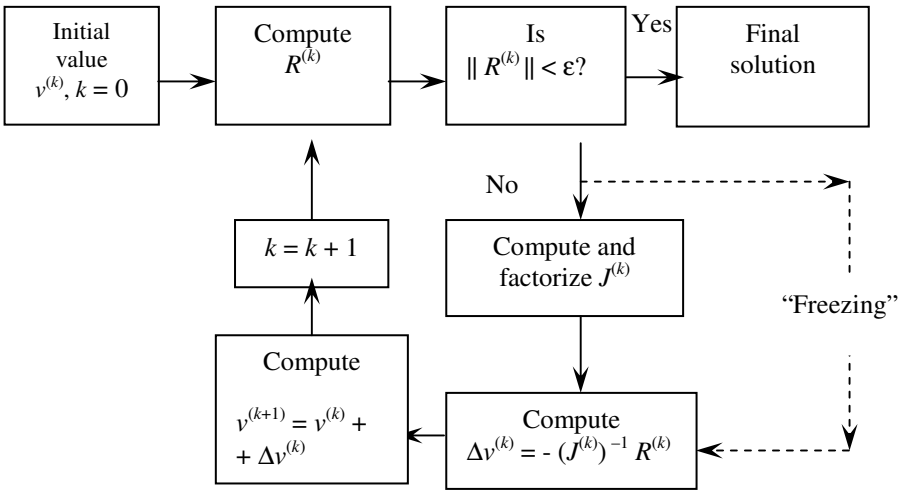


Fig. 3.2. Block scheme for Newton's method.

Convergence properties for Newton's method can be obtained based on the following. If

1. the norm of the inverse to the matrix $\underline{J}^{(0)}$ is bounded by a number a, i.e. $\left\| \left(\underline{J}^{(0)} \right)^{-1} \right\| \le a$;
2. the first vector norm of the difference $\Delta v^{(1)}$ is bounded by a number b, i.e. $\left\| \Delta \vec{v}^{(1)} \right\| = - \left\| - \left(\underline{J}^{(0)} \right)^{-1} \vec{R}^{(0)} \right\| \le b$;
3. the correction vector $\vec{R}$ has continuous second-order derivatives which satisfy the condition
$$\sum_{i,j=1}^{N} \left| \frac{\partial^2 R_m}{\partial v_i \partial v_j} \right| \le \frac{c}{N} \quad \text{for } \forall \vec{v}: \ \left\| \Delta \vec{v}^{(1)} \right\| < 2b.$$
4. the constants defined above satisfy the condition $abc < 0.5$;

then the solution by Newton's method, $\vec{v}^{(k)}$, converges to the exact solution $\vec{v}_c$, i.e.

$$\lim_{k\to\infty} \vec{v}^{(k)} = \vec{v}_c;$$

moreover,

$$\vec{R}\left(\vec{v}_c\right) = 0, \quad \left\|\vec{v}^{(k)} - \vec{v}_c\right\| \le \frac{b}{2^{k-1}}.$$

Remarks:

- *In the foregoing formulation, the following are used for the vector and matrix norms, respectively:* $\left\|\vec{v}\right\| = \max_i |v_i|, \quad \left\|\underline{J}\right\| = \max_i \left(\sum_{j=1}^{N} \left|J_{ij}\right| \right)$.
- *The proof of the convergence of Newton's method is approximately as complex as the solution of equation (3.1).*
- *The main difficulty with using Newton's method is due to the fact that the convergence radius decreases as N increases. This is why the value* $\vec{v}^{(0)}$ *for the initial iteration must be taken as close as possible to* $\vec{v}_c$.
- *Most of the computational time in Newton's method is expended on the computation of the matrix* $\underline{J}^{(k)}$ *and its factorization#. "Freezing"* $\underline{J}^{(k)}$ *for a few iterations, or for the whole computation cycle, may reduce this time, although in the latter case linear, rather than quadratic, convergence is obtained.*
- *This remark does not apply if a few initial Seidel iterations are carried out, or if under relaxation (*$\omega < 0.5$*) as in equation (3.6) is used. In this case, however, the convergence speed is substantially lower.*
- *The maximal effectiveness of Newton's method appears to be for small systems of strongly non-linear algebraic equations, e.g. full Navier-Stokes equations, Reynolds equations, Burgers' equation, etc. This is because the method is easily adapted to any arbitrary equation, as shown above.*

Matrix factorization in numerical linear algebra (NLA) typically serves the purpose of restating a given problem in such a way that it can be solved more readily, e.g. one major application is in the solution of linear systems of equations. The actual components of a factorization are now of prime importance as regards matrix factorization and the subsequent rank reduction of a matrix. We note in particular that several results available for many decades were recently rediscovered in the NLA literature.

3.2 Quasi-Newtonian Methods

The negative characteristics of Newton's method, i.e. small convergence radius, long computation time for the Jacobian $\underline{J}^{(k)}$ and the necessity to solve an LAEA for each iteration (factorization of $\underline{J}^{(k)}$) are easily overcome if the matrix $\underline{J}^{(k)}$ has some additional properties, e.g. if it is positive definite. In this case, equation (3.4) is replaced by

$$\vec{v}^{(k+1)} = \vec{v}^{(k)} - w^{(k)} \underline{H}^{(k)} \vec{R}^{(k)}, \tag{3.8}$$

where the matrix $\underline{H}^{(k)}$ is an approximation of $\left(\underline{J}^{(k)}\right)^{-1}$, which is systematically modified at each iteration so that it is tends to $\left(\underline{J}^{(k)}\right)^{-1}$. It is important to say that the modification of $\underline{H}^{(k)}$ is a substantially more economical process than the factorization of $\underline{J}^{(k)}$.

Now, rewrite equation (3.8) in the form

$$\vec{v}^{(k+1)} = \vec{v}^{(k)} - w^{(k)} \vec{\xi}^{(k)}. \tag{3.9}$$

The vector $\vec{\xi}^{(k)}$ may be treated as a solution direction. The scalar $w^{(k)}$ is chosen by the condition of a minimal norm $\left\|\vec{R}^{(k+1)}\right\|$ for the direction $\vec{\xi}^{(k)}$. Equation (3.9) provides a substantial increase in the convergence radius, as compared to Newton's method, for large N.

The Newton procedure is actually one of the most efficient and converges in only a few iterations. Newton linearization can be used to update coefficients iteratively and to provide a useful representation for the most non-linear problems arising in computational fluid dynamics (CFD). For example, some researchers have noted [2] that convergence of the iterations to update coefficients for streamwise steps in a boundary layer can be strongly accelerated by solving the coupled continuity and momentum equations.

However, the effectiveness of quasi-Newtonian methods depends also on additional properties, such as whether or not the matrix is positive definite. Their application has therefore to be analyzed separately. Mostly, their effectiveness is considered [2, 20] only together with an absolute minimization procedure. A full bibliography discussing the application of the quasi-Newtonian methods to some special problems can be found in the monograph of Fletcher [20]. In [20, 37], a number of algorithms for quasi-Newtonian methods are described.

3.3 Direct methods for the solution of a linear equation array

Let us consider the extended linear algebraic equation array (LAEA)

$$\underline{A}\vec{v} = \vec{b}\,, \tag{3.10}$$

where the components of matrix $\underline{A}$ and vector $\vec{b}$ are known. The methods of solution for equation (3.10) are separated into three classes depending on the form of matrix $\underline{A}$:

- $\underline{A}$ is compact (consists mainly of non-zero elements);
- $\underline{A}$ is sparse (consists mainly of zero elements);
- $\underline{A}$ is banded (the non-zero elements are located along lines or bands).

For the first class of methods, the most popular are the Gauss elimination method and factorization in tandem with the Gauss method. By factorization the matrix $\underline{A}$ transforms to

$$\underline{A} = \underline{L}\underline{V}\,, \tag{3.11}$$

where the matrices $\underline{L}$ and $\underline{V}$ are lower- and upper-triangular, respectively. Then, instead of (3.10), the following equation is solved:

$$\underline{V}\vec{v} = \underline{L}^{-1}\vec{b}. \tag{3.12}$$

Due to the structure of matrix $\underline{V}$, the process of solving (3.12) is reduced to consecutive substitutions for the calculation of $\underline{L}^{-1}\vec{b}$.

Remarks:

- *Compact matrices are obtained by discretization using the spectral[#] or panelling methods and the method of discrete singularities.*
- *Sparse matrices are also solved by the Gauss method, but instead of passing information about the entire matrix $\underline{A}$ only its non-zero elements and their location in the matrix $\underline{A}$ are stored (storage index $\underline{IA}$). The main problem here is that zero elements of the matrix $\underline{A}$ become non-zero after their elimination due to numerical solution. Therefore, for sparse matrices which have no banded structure, it is best to use the Householder method, Givens rotation or the QR-algorithm, which do not add non-zero elements to the matrix $\underline{A}$ [20].*

The application of three-point finite-difference schemes or finite elements with linear interpolation yields tridiagonal banded matrices. The utilization of finite-difference or finite-element schemes of higher order gives banded structure matrices $\underline{A}$ with bandwidth greater than 3.

3.3.1 Thomas algorithm

This is the most popular algorithm for the solution of linear algebraic equation arrays

[#] During the last three decades, spectral methods have emerged as successful, and often superior, alternatives to the better-known finite-difference and finite-element methods. Applications include several key areas of computational fluid dynamics, the modelling of different types of wave motion, weather forecasting, etc. The main ingredients of spectral approximations are Fourier and Chebyshev interpolants, their spectral accuracy, the role of aliasing, differentiation matrices, etc. For time-dependent problems, the energy balance for the linear wave and heat equations and the phenomena of dissipation and dispersion for non-linear Korteweg-de Vries (KDV) and Navier-Stokes equations are modelled by spectral methods. In particular, the questions of the stability and convergence of spectral and other methods have been comprehensively addressed in the literature.

(LAEA) such as (3.13) and the best method for block tridiagonal matrices. The modified Thomas algorithm is used for block pentadiagonal matrices, and matrices having tridiagonal and pentadiagonal blocks in turn. In addition, matrix marching is also used.

Consider the matrix equation

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & 0 & \\ & \vdots & & & & & \\ & a_i & b_i & c_i & & & \\ & & & \vdots & & & \\ 0 & & & a_{N-1} & b_{N-1} & c_{N-1} & \\ & & & a_N & b_N & & \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_{N-1} \\ v_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_i \\ \vdots \\ d_{N-1} \\ d_N \end{bmatrix}. \tag{3.13}$$

The Thomas algorithm consists of two steps, as shown in Fig. 3.3. On the first step, the following equations are used to transform the matrix to a simpler form (in order to eliminate the lower diagonal and to obtain ones on the main diagonal):

$$c'_1 = \frac{c_1}{b_1}, \quad d'_1 = \frac{d_1}{b_1}, \quad c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}, \quad d'_i = \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} \quad (i = \overline{2, N}) \tag{3.14}$$

Equation array (3.13) is transformed to

$$\begin{bmatrix} 1 & c'_1 & & & & & \\ & 1 & c'_2 & & & 0 & \\ & & \cdots & & & & \\ & & & 1 & c'_i & & \\ & & & & \ddots & & \\ 0 & & & & 1 & c'_{N-1} & \\ & & & & & 1 & \end{bmatrix} \begin{bmatrix} v_1 \\ \cdot \\ \cdot \\ \cdot \\ v_i \\ \cdot \\ v_N \end{bmatrix} = \begin{bmatrix} d'_1 \\ \cdot \\ \cdot \\ \cdot \\ d'_i \\ \cdot \\ d'_N \end{bmatrix}. \tag{3.15}$$

Marching backwards then gives

$$v_N = d'_N,\ v_i = d'_i - v_{i+1}c'_i\ \ (i = \overline{N-1,\ 1}), \tag{3.16}$$

which is obtained by considering the solution of (3.15), starting with the last equation.

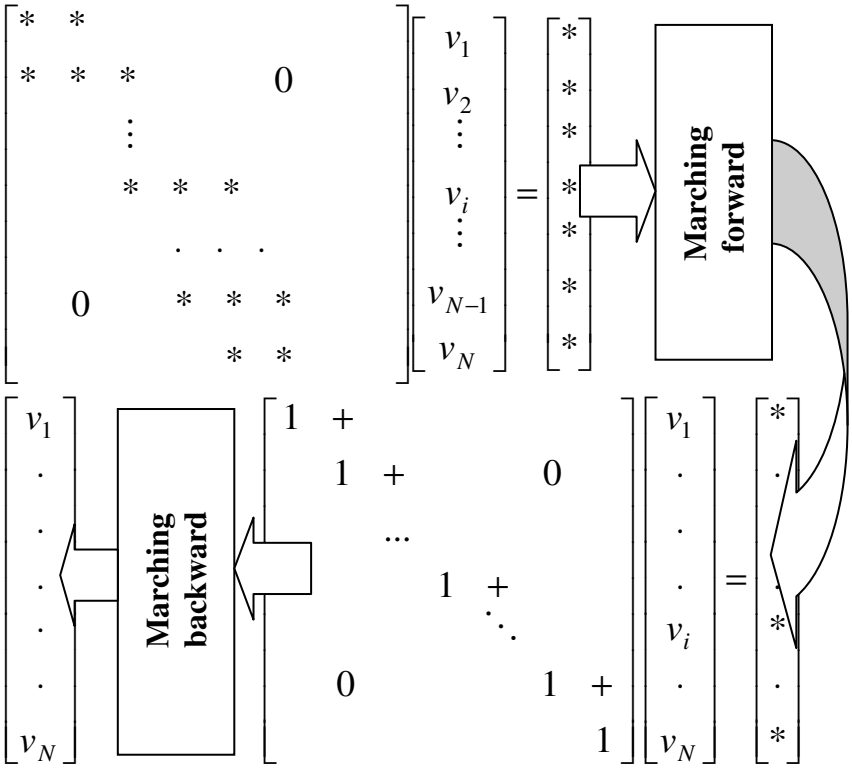


Fig. 3.3. A schematic of the Thomas algorithm.

Consequently, the Thomas algorithm is economical, requiring just $5N-4$ elementary operations.

Now, let us consider a matrix equation with non-zero elements in a band of five diagonals $\{e_i, a_i, b_i, c_i, f_i\}$. The modified marching algorithm consists of three steps and is similar to the one considered above. The first step of the algorithm leads to the elimination of the first lower diagonal e_i, which transforms the equation array to

$$\begin{bmatrix} b_1 & c_1 & f_1 & & & & & \\ a_2 & b_2 & c_2 & f_2 & & & 0 & \\ & a_3' & b_3' & c_3' & f_3' & & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & a_i' & b_i' & c_i' & f_i' & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & 0 & & & a_{N-1}' & b_{N-1}' & c_{N-1}' & \\ & & & & & a_N' & b_N' & \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ \cdot \\ \cdot \\ \cdot \\ v_i \\ \cdot \\ \cdot \\ v_N \end{bmatrix} = \begin{bmatrix} d_1 \\ \cdot \\ \cdot \\ \cdot \\ d_i' \\ \cdot \\ \cdot \\ d_N \end{bmatrix}, \tag{3.17}$$

where

$$\begin{aligned} &a_i' = a_i - \frac{e_i b_{i-1}'}{a_{i-1}'}, \quad b_i' = b_i - \frac{e_i c_{i-1}'}{a_{i-1}'}, \quad f_i' = f_i, \\ &c_i' = c_i - \frac{e_i f_{i-1}'}{a_{i-1}'}, \quad d_i' = d_i - \frac{e_i d_{i-1}'}{a_{i-1}'}, \quad i = 3, N. \end{aligned} \tag{3.18}$$

Thus, in the original LAEA with five diagonals, the first lower diagonal e_i has already been eliminated and we have obtained an equation array, (3.17), with a tetradiagonal matrix. In the second step, the elements a_i' are eliminated, yielding:

$$\begin{bmatrix} 1 & c_1'' & f_1 & & & & & \\ & 1 & c_2'' & f_2'' & & 0 & & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & & 1 & c_i'' & f_i'' & \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & 0 & & & 1 & c_{N-1}'' & \\ & & & & & & 1 & \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ \cdot \\ \cdot \\ \cdot \\ v_i \\ \cdot \\ \cdot \\ v_N \end{bmatrix} = \begin{bmatrix} d_1 \\ \cdot \\ \cdot \\ \cdot \\ d_i'' \\ \cdot \\ \cdot \\ d_N'' \end{bmatrix}, \tag{3.19}$$

where

$$c_i'' = \frac{c_i' - a_i' f_{i-1}''}{b_i' - a_i' c_{i-1}''}, \; d_i'' = \frac{d_i' - a_i' d_{i-1}''}{b_i' - a_i' c_{i-1}''}, \; f_i'' = \frac{f_i'}{b_i' - a_i' c_{i-1}''}, \quad i = 2,3,..,N. \tag{3.20}$$

In the third step, the solution is calculated by the following formula:

$$v_i = d_i'' - c_i'' v_{i+1} - f_i''\ v_{i+2}. \tag{3.21}$$

Remarks:

- *Matrix $\underline{A}$ must, for the most part, be block diagonal in order to avoid ill-conditioning for the Thomas algorithm:*

$$|b_i| > |a_i| + |c_i|. \tag{3.22}$$

For the modified Thomas algorithm, matrix $\underline{A}$ should satisfy the following condition:

$$|b_i| > |a_i| + |c_i| + |e_i| + |f_i|. \tag{3.23}$$

- *The different steps of the modified algorithm can be considered as several operations (direct marching) that are required to transform the matrix $\underline{A}$ to upper-triangular form. After that, the vector $\vec{v}$ may be calculated by a normal backward substitution. Evidently, this algorithm can be applied to a system with a wider band, but the advantages decrease as compared, for example, with the Gauss method.*

The Thomas algorithm can also be used for more complex cases than solving linear algebraic equation arrays (LAEAs) with band matrices, e.g. for LAEAs with block tridiagonal matrices (3.24), where $\underline{a}_i, \underline{b}_i, \underline{c}_i$ are sub-matrices of dimension $M \times M$, and $\vec{v}_i, \vec{d}_i$ are *M*-component sub-vectors. The number *M* is associated with the number of the equations written at each point of the grid (for example, for 3-D viscous flow $M = 5$). Thus, $\vec{v}_i$ is a sub-vector of the solution corresponding to a particular grid point, and matrix equation (3.24) is then a system consisting of *N* blocks of equations. Each of these (for a particular grid point), in turn, consists of *M* equations.

$$\begin{bmatrix} \underline{b}_1 & \underline{c}_1 & & & & \\ \underline{a}_2 & \underline{b}_2 & \underline{c}_2 & & 0 & \\ & \vdots & & & & \\ & \underline{a}_i & \underline{b}_i & \underline{c}_i & & \\ & & & \vdots & & \\ 0 & & & \underline{a}_{N-1} & \underline{b}_{N-1} & \underline{c}_{N-1} \\ & & & \underline{a}_N & \underline{b}_N & \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_i \\ \vdots \\ \vec{v}_{N-1} \\ \vec{v}_N \end{bmatrix} = \begin{bmatrix} \vec{d}_1 \\ \vec{d}_2 \\ \vdots \\ \vec{d}_i \\ \vdots \\ \vec{d}_{N-1} \\ \vec{d}_N \end{bmatrix}, \tag{3.24}$$

Equation array (3.24) is solved by a procedure similar to the Thomas algorithm. First, the tridiagonal matrix of the blocks from (3.24) is transformed to upper-triangular form by the elimination of the sub-matrices $\underline{a}_i$. In the same way as for (3.14), the first block of equations yields

$$\underline{c}'_1 = \left(\underline{b}_1\right)^{-1} \underline{c}_1 \ , \ \vec{d}'_1 = \left(\underline{b}_1\right)^{-1} \vec{d}'_1, \tag{3.25}$$

and, for a block of general form, we obtain

$$\underline{b}'_i = \underline{b}_i - \underline{a}_i \ \underline{c}'_{i-1} \ , \ \underline{c}'_i = \left(\underline{b}_i\right)^{-1} \underline{c}_1 \ , \ \vec{d}'_i = \left(\underline{b}_i\right)^{-1} \left\{\vec{d}_i - \underline{a}_i \ \vec{d}'_{i-1}\right\}. \tag{3.26}$$

Equations (3.25) and (3.26) have the explicit inverse matrices. In practice, it is more economical to solve the separate *M*-component sub-systems instead of computing the inverse matrices, e.g. the equation

$$\underline{b}'_i \underline{c}'_i = \underline{c}_i \tag{3.27}$$

is solved with respect to $\underline{c}'_i$. After computation by (3.25) and (3.26), the system (3.24) is transformed to upper-triangular form with a replacement of $\underline{c}_i, \vec{d}_i$ by $\underline{c}'_i, \vec{d}'_i$, respectively, and the replacement of sub-matrices $\underline{b}_i$ by identity matrices $\underline{E}$.

The second step, using (3.16), requires the backward substitutions:

$$\vec{v}_N = \vec{d}'_N \,,\ \vec{v}_i = \vec{d}'_i - \underline{c}'_i \vec{v}_{i+1} \,. \tag{3.28}$$

Remark: *The Thomas block algorithm requires around $5NM^3/3$ operations, while the Gauss elimination procedure requires considerably more: $(NM)^3/3$.*

3.3.2 Orthogonalization

The orthogonalization methods (see, for example [25]) include the Jacobi rotation method, which is used also to calculate eigenvalues for symmetric matrices, and similar methods which are applied for arbitrary matrices: Householder method, Givens method and the QR-algorithm [45].

These methods are based on the idea of obtaining zero elements under the leading diagonal. For this purpose, orthogonal matrices M (satisfying the condition: $M^T \equiv M^{-1}$) multiply the matrix $\underline{A}$. The price for a substantial increase of numerical stability is a higher arithmetic cost. Orthogonalization methods were developed for the computation of eigenvectors and eigenvalues of a matrix $\underline{A}$. The possibility to use them for the solution of LAEAs (3.13) is important if $\underline{A}$ is a sparse matrix. Orthogonalization is especially valuable in the Givens rotation and the Householder method. Despite a much higher computational cost as compared to the Gauss method (for compact $\underline{A}$), their undoubted advantages are that:

- the character of the calculations is local;
- they can be parallelized.

We begin by considering the solution of the full eigenvalue problem by the rotation method for real symmetric matrices (suitable also for arbitrary Hermitian matrices). Define the closeness measure of a matrix $\underline{A}$ to diagonal form by

$$\sigma_i(\underline{A}) = \sum_{j=1,\, j\neq i}^{N} \left|a_{ij}\right|^2 \quad (i = \overline{1,N}),$$

and introduce

$$t(\underline{A}) = \sigma_1 + \sigma_2 + \ldots + \sigma_N = \sum_{i\neq j} \left|a_{ij}\right|^2 .$$

Definition: The construction procedure of a matrix sequence $\left\{\underline{A}^k\right\}$, $\underline{A}^0 \equiv \underline{A}$ $(k = 0,1,2,\ldots)$ is monotonic if

$$t\left(\underline{A}^k\right) < t\left(\underline{A}^{k-1}\right) . \tag{3.29}$$

As is well-known from courses on linear algebra, an arbitrary real symmetric matrix $\underline{A}$ can be reduced to diagonal form by the following procedure:

$$\underline{A} = \underline{U}^{-1}\underline{\Lambda}\underline{U} , \tag{3.30}$$

where $\underline{U}$ is an orthogonal matrix: $\underline{U}\underline{U}^{-1} = \underline{E}$, $\underline{E}$ is the identity matrix and $\underline{\Lambda}$ is a diagonal matrix containing the eigenvalues λ_i, $i = \overline{1,N}$ of matrix $\underline{A}$. Because $\underline{U}^{-1} \equiv \underline{U}^T$ for a real symmetric matrix $\underline{A}$, (3.30) is equivalent to

$$\underline{U}^T \underline{A}\underline{U} = \underline{\Lambda} . \tag{3.31}$$

To use (3.31), it is necessary to build a sequence of orthogonal transformations, which would allow an unlimited reduction in the moduli of $\underline{A}$'s off-diagonal elements.

The algorithm for the Jacobi rotation method consists of the construction of a matrix sequence $\left\{\underline{A}^k\right\}$, such that

$$\underline{A}^{k+1} = \underline{U}^T{}_{ij}(\varphi)\underline{A}^k\underline{U}_{ij}(\varphi), \tag{3.32}$$

where $\underline{U}_{ij}(\varphi)$ are the orthogonal matrices of the plane rotations (3.33) that are made for an arbitrary angle φ, i.e. $\underline{U}_{ij}^{-1}(\varphi)\underline{U}_{ij}(\varphi) = \underline{E}$, where

$$\underline{U}_{ij}(\varphi) = \begin{bmatrix} 1 & & & & & \\ & \ddots & & & 0 & \\ & & 1 & & & \\ & & & \cos\varphi \quad \dots \quad -\sin\varphi & & \\ & & & \sin\varphi \quad \dots \quad \cos\varphi & & \\ & & & & 1 & \\ & 0 & & & & \ddots \\ & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ (i) \\ \\ (j) \\ \\ \\ \end{matrix} . \qquad (3.33)$$

$$(i) < (j)$$

Let $\underline{A}^k = \left\{a_{ij}^k\right\}$ and let $a_{ij}^k, i < j$ be the non-diagonal element of the matrix $\underline{A}^k$ which possesses the maximum modulus. Using the rotation matrices $\underline{U}_{ij}^k = \underline{U}_{ij}\left(\varphi^k\right)$, construct the sequence

$$\underline{A}^{k+1} = \left[\underline{U}_{ij}^k\right]^T \underline{A}^k \underline{U}_{ij}^k, \qquad (3.34)$$

and build the matrix $\underline{B}^k = \left\{b_{ij}^k\right\}$, where $\underline{B}^k = \underline{A}^k U_{ij}^k$. According to the definition of $\underline{U}_{ij}^k$, all the columns of matrix $\underline{B}^k$, except for the i-th and j-th, coincide with the corresponding columns of the matrix $\underline{A}^k$. The elements of columns with numbers (i) and (j) are computed by the equations:

$$\begin{aligned} b_{ni}^k &= a_{ni}^k \cos\varphi^k + a_{nj}^k \sin\varphi^k, \\ b_{nj}^k &= -\ a_{ni}^k \sin\varphi^k + a_{nj}^k \cos\varphi^k, \ (n = \overline{1,N}) \end{aligned} \qquad (3.35)$$

In a similar way, all the rows of matrix $\underline{A}^{k+1}$, except (i) and (j), coincide with the corresponding rows of the matrix $\underline{B}^k$. The elements of the rows with numbers (i) and (j) are computed by the equations:

$$
\begin{aligned}
a_{in}^{k+1} &= b_{in}^{k}\cos\varphi^{k} + b_{jn}^{k}\sin\varphi^{k}, \\
a_{jn}^{k+1} &= -b_{in}^{k}\sin\varphi^{k} + b_{jn}^{k}\cos\ \varphi^{k},\ (n=\overline{1,N}).
\end{aligned}
\tag{3.36}
$$

Now, from (3.35) and (3.36), it follows that

$$
\begin{aligned}
&a_{ij}^{k+1} = b_{ij}^{k}\cos\varphi^{k} + b_{jj}^{k}\sin\varphi^{k} = \\
&= \left(a_{ij}^{k}\cos\varphi^{k} - a_{ii}^{k}\sin\varphi^{k}\right)\cos\varphi^{k} + \left(a_{jj}^{k}\cos\varphi^{k} - a_{ji}^{k}\sin\varphi^{k}\right)\sin\varphi^{k}
\end{aligned}
\tag{3.37}
$$

For a symmetric matrix, $a_{ij}^{k} \equiv a_{ji}^{k}$. Furthermore, equation (3.37) simplifies to

$$
a_{ij}^{k+1} = a_{ij}^{k}\cos 2\varphi^{k} + \frac{1}{2}\left(a_{jj}^{k} - a_{ii}^{k}\right)\sin 2\varphi^{k}. \tag{3.38}
$$

Choosing φ^{k} for the case where $a_{ij}^{k+1} \equiv 0$ yields

$$
\tan 2\varphi^{k} = \frac{2a_{ij}^{k}}{a_{ii}^{k} - a_{jj}^{k}}. \tag{3.39}
$$

Remarks:

- *An infinite number of procedures similar to (3.34) can be built satisfying the condition (3.35).*
- *The reduction of a matrix to a diagonal form using the Jacobi method requires an infinitely large number of steps, because new non-zero elements appear at locations that previously contained zeros.*

The main difference between the Givens method [25] and the Jacobi method lies in the construction of the algorithms. The algorithm of the Jacobi rotation method considered above stores the non-zero elements obtained at previous steps for further calculations. But, in accordance with the Givens method, a symmetric matrix is transformed only to block tridiagonal form, whilst a non-symmetric one is transformed to Hessenberg form,

which is block-triangular with an additional non-zero diagonal below the main diagonal in an otherwise zero block domain. Let us now analyze the difference between these methods in detail.

In the case of symmetric matrices, the Householder method [25] allows reduction of a matrix to a tridiagonal form in nearly half the number of operations that are required by the Givens method. The reason is that, by this method, all the elements, which do not belong to the three main diagonals, become zero. The advantage of the method is overwhelming for huge sparse matrices due to the elimination of a number of needless computations involving zero matrix elements. With the Householder method involving $(N-2)$ main steps, the following transformations are applied:

$$\underline{A}^{k+1} = \left[\underline{P}_{ij}^{k}\right]^{T} \underline{A}^{k} \underline{P}_{ij}^{k}. \tag{3.40}$$

Here, each transformation matrix has the form

$$\underline{P}^{k} = \underline{E} - \frac{\underline{X}^{k}\left[\underline{X}^{k}\right]^{T}}{2K_{k}^{2}}, \tag{3.41}$$

where

$$\begin{cases} x_{ik} = 0\ ,\ i = 1,2,\ldots,k\ ; \\ x_{ik} = a_{ki}\ ,\ i = k+2,\ldots,N; \\ x_{k+1,\,k} = a_{k,\,k+1} \mp S_{k}. \end{cases} \tag{3.42a}$$

$$S_{k} = \left[\sum_{i=k+1}^{N} a_{k\,i}^{2}\right]^{1/2},\ 2\ K_{k}^{2} = S_{k}^{2} \mp a_{k\,,\,k+1}\ S_{k}\,. \tag{3.42b}$$

In (3.42), the signs are chosen to correspond with the sign of the element $a_{k,k+1}$, allowing us to obtain the maximal value of $x_{k,k+1}$.

Remarks:

- *Using the Givens and Householder methods for the solution of the linear algebraic equation array (LAEA) of a sparse symmetric matrix, the reduced block tridiagonal matrix of LAEA is solved using the QR-algorithm or the Thomas method.*
- *Although the Householder method uses the more complex Hermitian matrix instead of the plane rotation matrix (3.33), the result is the same for both the eigenvalue problem, as well as the solution of the LAEA with a sparse matrix.*

Now, let us consider the solution of a full eigenvalue problem for an arbitrary real quadratic matrix $\underline{A}$, using the QR-algorithm to demonstrate its characteristics.

Lemma (Bakhvalov, et. al. [4]): An arbitrary quadratic matrix can be presented as a product of an orthogonal and right-hand triangular matrix.

As a result of the lemma, the matrix $\underline{A}$ can be written in the form

$$\underline{A} = \underline{Q}_1 \underline{R}_1 \ , \tag{3.43}$$

where $\underline{Q}_1$ and $\underline{R}_1 \equiv \underline{A}^{k-1}$ are orthogonal and right-hand triangular matrices with respect to the matrix $\underline{A}$, respectively. Multiplying (3.43) from the right with $\underline{Q}_1^{-1}$ yields

$$\underline{R}_1 = \underline{Q}_1^{-1} \underline{A} \ . \tag{3.44}$$

Now substituting (3.44) into (3.43), one can obtain from $\underline{A}$ its similar matrix, $\underline{A}^1$, given by

$$\underline{A}^1 = \underline{Q}_1^{-1} \underline{A} \underline{Q}_1 \ . \tag{3.45}$$

Building a sequence of matrices $\left\{ \underline{A}^k \right\}$ by the rule

$$\underline{A}^{k+1} = \underline{Q}_{k+1}^{-1} \underline{A}^k \underline{Q}_{k+1} \ , \tag{3.46}$$

it can be proved that a procedure such as (3.46) transforms the matrix $\underline{A} \equiv \underline{A}^0$ to

$$\underline{A} = \underline{Q}\underline{\Lambda}\underline{Q}^{-1} , \tag{3.47}$$

where $\underline{Q} = \underline{Q}_1 \ \underline{Q}_2 \ \cdots \ \underline{Q}_N$, and $\underline{\Lambda} = \{\lambda_{ij}\}$ is the right Jordan canonical form, where $\lambda_{ij} \equiv 0$ for $j < i$ and $\lambda_{ii} \equiv \lambda_i$ ($j > i+1$) are the eigenvalues of matrix $\underline{A}$, and $\lambda_{i,\, i+1}$ is 0 or 1. For further details, see monograph [4], which considers several different algorithms for the computation of a sequence of the orthogonal matrices $\{\underline{Q}_k\}$ and their respective computational costs.

Remarks:

- *By using the Givens and Householder methods for solving the eigenvalue problem for a non-symmetric matrix, a matrix reduced to Hessenberg form is transformed further by the QR-algorithm.*
- *By using the QR-algorithm for the solution of an LAEA for a sparse matrix, it is recommended that, on the first step, the Givens method or Householder method, which transform an arbitrary matrix to Hessenberg form, be used. Finding a QR-expansion is simpler for a matrix in Hessenberg form. The transformation applied on matrix $\underline{A}$ should also be used on vector $\vec{b}$.*

3.4 Iterative methods for linear equation arrays

For large numbers of internal points, direct methods for the solution of LAEAs cannot be used to due to the increasing inaccuracy of the approximations, caused by the huge number of the arithmetic operations required. Iterative methods help to avoid these complications. They allow us to obtain sequence of vectors from $\vec{v}^{(0)}$ to $\{\vec{v}^{(m)}\}$, the limit of which is the solution of the equation array, i.e.

$$\lim_{m \to \infty} \vec{v}^{(m)} = \vec{v}_c . \tag{3.48}$$

The method is said to converge if (3.48) is satisfied for arbitrary $\vec{v}^{(0)}$.

For all the methods presented above, consider a matrix $\underline{A}$ in the form

$$\underline{A} = \underline{M} - \underline{N} \ , \tag{3.49}$$

where $\| \underline{M} \| \approx \| \underline{A} \|$, and is such that it may be easily factored numerically, e.g. $\underline{M}$ may be block tridiagonal. This transforms vector equation (3.2) to the following form:

$$\underline{M}\vec{v}^{(m+1)} = \underline{N}\vec{v}^{(m)} + \vec{b} \ . \tag{3.50}$$

The formal solution of (3.50) is

$$\vec{v}^{(m+1)} = \underline{M}^{-1}\underline{N}\vec{v}^{(m)} + \underline{M}^{-1}\vec{b} \tag{3.51}$$

or

$$\vec{v}^{(m+1)} = \vec{v}^{(m)} - \underline{M}^{-1}\vec{R}^{(m)} , \tag{3.52}$$

where

$$\vec{R}^{(m)} = \underline{A}\vec{v}^{(m)} - \vec{b} \tag{3.53}$$

is the discrepancy vector.

It is well-known that the numerical schemes (3.51), (3.52) guarantee convergence to the exact solution if the spectral radius (the maximal eigenvalue) of the matrix $\underline{M}^{-1}\underline{N}$ is less than unity, provided that the matrices $\underline{M}$ and $\underline{N}$ are chosen appropriately.

Remark: *in addition, the solution by iterative methods is simplified due to the use of the same matrices at each step of the solution!*

Let us consider the popular Gauss-Seidel algorithm and some of its variants. Write the matrix $\underline{A}$ in the form

$$\underline{A} = \underline{D} - \underline{G} - \underline{F}, \tag{3.54}$$

where

$$\underline{D} = \left\{ d_{ij} \right\} : d_{ii} \equiv a_{ii} \ , \ d_{ij} \equiv 0 \left(i \neq j \right),$$

$$\underline{G} = \left\{ g_{ij} \right\} : g_{ij} \equiv -a_{ij} \ \ (i > j) \ ; \ g_{ij} \equiv 0 \ \ (i \leq j)$$

and $\underline{F} = \left\{ f_{ij} \right\} : f_{ij} \equiv -a_{ij} \ \ (i < j) \ ; \ f_{ij} \equiv 0 \ \ (i \geq j)$ are block diagonal, lower-triangular and upper-triangular matrices, respectively. Suppose that all diagonal elements are non-zero, so that $a_{ii} \neq 0$, and compare the three classic methods:

- Jacobi method: $\underline{M} = \underline{D}, \ \underline{N} = \underline{G} + \underline{F}$,

$$\underline{D} \vec{v}^{(m+1)} = \vec{b} + \left(\underline{G} + \underline{F} \right) \vec{v}^{(m)} . \tag{3.55}$$

- Gauss-Seidel method: $\underline{M} = \underline{D} - \underline{G}$ and $\underline{N} = \underline{F}$,

$$\left(\underline{D} - \underline{G} \right) \vec{v}^{(m+1)} = \vec{b} + \underline{F} \vec{v}^{(m)}; \tag{3.56}$$

- Successive overrelaxation (SOR): $\underline{M} = \frac{1}{w} \underline{D} - \underline{G}, \ \underline{N} = \left(\frac{1}{w} - 1 \right) \underline{D} + \underline{F}$

$$\left(\underline{D} - w \underline{G} \right) \vec{v}^{(m+1)} = w \vec{b} + \left(1 - w \right) \underline{D} \vec{v}^{(m)} + w \underline{F} \vec{v}^{(m)} . \tag{3.57}$$

Here, w is the relaxation parameter.

The Jacobi method is the most economical of these since it solves the LAEA at each step with a block diagonal matrix. In the Gauss-Seidel method, at each step, a block triangular matrix is solved. The Jacobi method, however, requires twice as much computer memory as the Gauss-Seidel method. The successive overrelaxation method is obtained by modifying the Gauss-Seidel method, thereby accelerating its convergence.

Convergence of the iterative methods:

- if $\underline{A}$ is a symmetric and positive definite matrix, then the Gauss-Seidel and successive overrelaxation methods converge for $0 < w \le 2$;
- if the matrix $\underline{A}$ is diagonally dominant, so that

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| , \tag{3.58}$$

then the Jacobi and Gauss-Seidel methods converge.

Remarks:

- *The Jacobi and Gauss-Seidel method converge linearly. If $w = w_{opt}$, then the SOR method has nearly quadratic convergence. However, the computation of w_{opt} requires a certain number of operations, which is approximately the same as for the solution of an equation array because*

$$w_{opt} = 2 \Big/ \left(1 + \left(1 - \mu^2\right)^{\frac{1}{2}}\right), \tag{3.59}$$

where μ is the maximal eigenvalue for the expression $\underline{E} - \underline{D}^{-1}\underline{A}$ (where $\underline{E}$ is identity matrix). One possible strategy is to make w_{opt} more exact by using (3.59) iteratively, if the eigenvalue μ can somehow be estimated in advance.

- *The iterative methods allow the use of acceleration procedures, such as the Chebyshev method or the method of conjugate gradients[#]. Unlike SOR, it is not necessary to choose parameters empirically.*

[#] Conjugate Gradient Method (CGM)
Newton's method converges faster (quadratically) than the method of steepest descent (linear convergence rate) because it uses more information about the function *f* being minimized. Steepest descent approximates the function locally with planes, because it only uses gradient information. All it can do is to go downhill. Newton's method approximates *f* with paraboloids, and then jumps at every iteration to the lowest point of the current approximation. The bottom line is that fast convergence requires work equivalent to evaluating the Hessian of *f*. *Prima facie*, the method of conjugate gradients seems to violate this principle: it achieves fast, superlinear convergence, similar to Newton's method, but it only requires gradient information. This paradox, however, is only apparent. Conjugate gradients works by taking *n* steps for each of the steps in Newton's method. It solves the linear system of Newton's method effectively, but it does so by a sequence of *n* one-dimensional minimisations, each requiring one gradient computation and one line search. Overall, the work done by conjugate gradients is equivalent to that done by Newton's method. However, the system is never constructed explicitly, and the matrix is never stored. This is very important in the case of thousands or even

The well-known conjugate gradient method[#] uses the following steps:

$$
\begin{aligned}
&1)\vec{v}^{(m+1)} = \vec{v}^{(m)} + \lambda^{(m)}\vec{P}^{(m)}, \Rightarrow \qquad 2)\vec{R}^{(m+1)} = \vec{R}^{(m)} - \lambda^{(m)}\vec{U}^{(m)}, \Rightarrow \\
&3)\rho^{(m+1)} = \left(\vec{R}^{(m+1)}\right)^T \vec{R}^{(m+1)}, \Rightarrow \qquad 4)\alpha^{(m+1)} = \frac{\rho^{(m+1)}}{\rho^{(m)}}, \Rightarrow \\
&5)\vec{P}^{(m+1)} = \vec{R}^{(m+1)} + \alpha^{(m+1)}\vec{P}^{(n)}, \Rightarrow \qquad 6)\vec{U}^{(m+1)} = \underline{A}\ \ \vec{P}^{(m+1)}, \Rightarrow \\
&7)\lambda^{(m+1)} = \frac{\rho^{(m+1)}}{\left(\vec{P}^{(m+1)}\right)^T \vec{U}^{(m+1)}},
\end{aligned}
\tag{3.60}
$$

[#] Conjugate Gradient Method (CGM) (continued from the previous page):

millions of components. These high-dimensional problems arise typically from the discretization of a PDE, e.g. if computing the motion of points in an image as a consequence of camera motion. PDEs relate image intensities over space and time to the motion of the underlying image features. At every pixel in the image, a vector represents this motion, called the motion field, whose magnitude and direction describe the velocity of the image feature at that pixel. Thus, if an image has, say, a quarter of a million pixels, there are n=500,000 unknown motion field values. Storing and inverting a **500,000x500,000** Hessian is out of the question. In cases like these, conjugate gradients saves the day.

CGMs are typified by the so-called Polak-Ribière variation. It is introduced in three steps. First, it is developed for the simple case of minimising a quadratic function $f(x)$ with a positive-definite and known Hessian. In this case, minimisation is equivalent to solving a linear system. Rather than an iterative method, CGM is a direct method for the quadratic case. This means that the number of iterations is fixed. Specifically, the method converges to the solution in n steps, where n is the number of components of x. Due to equivalence with a linear system, CGM for the quadratic case can also be seen as an alternative method for solving a linear system. Second, the assumption that the Hessian is known is removed. This is the main reason for using conjugate gradients. Third, the CGM may be extended to general functions $f(x)$. In this case, the method is no longer direct, but iterative, and the cost of finding the minimum depends on the desired accuracy. This occurs because the Hessian of f is not constant, as it was in the quadratic case. Consequently, a certain property that holds in the quadratic case is now valid only approximately. In spite of this, the convergence rate of CGM is superlinear, lying somewhere between Newton's method and steepest descent. Finding tight bounds for the convergence rate of conjugate gradients is hard, and the proof is omitted here; we rely instead on the intuition that CGM solves the system, and that the quadratic approximation becomes more and more valid as the algorithm converges to the minimum. If the function f starts to behave like a quadratic function early on, that is, if f is nearly quadratic in a large neighbourhood of the minimum, convergence is fast, as it requires close to the n steps that are necessary in the quadratic case, and each of the steps is simple. This combination of fast convergence, modest storage requirements, and low computational cost per iteration explains the popularity of conjugate gradients methods for the optimization of functions of a large number of variables.

The algorithm (3.60) is an economical one, requiring only one matrix-vector multiplication: step 6. Vector $\vec{P}^{(m)}$ determines a search direction, and $\vec{P}^{(0)} = \vec{R}^{(0)}$. It is important to note that the discrepancy vector $\vec{R}^{(m)}$ is calculated recursively and not by definition (3.53). A further elaboration of the conjugate gradients' method is available for speeding-up its convergence, as well as other iterative and multigrid methods, are considered by Fletcher [20].

3.5 Pseudo-nonstationary method

The alternative to solving for the algebraic equation array, which appears from the discretization of a PDE for a stationary problem, is the construction of an equivalent non-stationary problem and to solve it using a marching method until the stationary regime is reached. In such pseudo-nonstationary procedures, time is considered as an iterative parameter. We consider a few examples. Discretization of the Laplace equation,

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0 \tag{3.61}$$

by using $\Delta x = \Delta y$, a central-difference scheme and the Jacobi method gives

$$v_{j,k}^{(m+1)} = 0.25\left(v_{j,k+1}^{(m)} + v_{j,k-1}^{(m)} + v_{j+1,k}^{(m)} + v_{j-1,k}^{(m)} \right) . \tag{3.62}$$

The discretization of the equivalent non-stationary equation

$$\frac{\partial v}{\partial t} = \alpha\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{3.63}$$

with $\Delta x = \Delta y$ results in

$$v_{j,k}^{n+1} = (1-4s)\ \ v_{j,k}^{n} + s\left(v_{j,k+1}^{n} + v_{j,k-1}^{n} + v_{j+1,k}^{n} + v_{j-1,k}^{n}\right), \tag{3.64}$$

where $s = \alpha\Delta t \big/ (\Delta x)^2$. If $s = 0.25$ in (3.64), then we obtain (3.62). This demonstrates the link between the non-stationary formulation of the problem and the iteration method. An important advantage of the non-stationary statement of the problem is the possibility of splitting a solution into *OX* and *OY*. The splitting procedure will be presented when the multidimensional heat conductivity equation is considered. Because the non-stationary solution is not of much interest in this case, the time step sequence may be chosen in such a way as to obtain the stationary solution as quickly as possible.

In circumferential flow problems, one often encounters the situation where the stationary regime is achieved much faster in some numerical domains than in others. Therefore, due to such a variation in the speed of convergence in space, it is reasonable to modify (3.63) by introducing an additional coefficient in front of the time derivative, so that

$$c(x,y)\frac{\partial v}{\partial t} = \alpha\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right). \tag{3.65}$$

Here the function $c(x,y)$ is chosen so that a speed of convergence in the entire numerical domain is uniform.

Remarks:

- *The advantages of the pseudo-nonstationary method are the following: large radius of convergence; economical iterations; low memory requirement; the possibility to use as, an initial guess, any arbitrary approximate solution and to reveal in the process of solution any non-stationary properties of the problem solved, which is difficult to do a priori.*
- *A big disadvantage is the slow convergence rate of the pseudo-nonstationary method, as compared to the iterative methods.*

3.6 Strategic approaches for solving stationary problems

Most problems in continuum mechanics can be solved using the following strategic approaches:

- Look for solution of the given problem with respect to a potential φ and stream function ψ, or with respect to a pressure p. As a rule, one needs to solve an equation with a matrix having block diagonal dominance; this equation is often even a linear one, which encourages the use of iterative methods.

- If the substitutive equations contain convective non-linear terms, then diagonal dominance appears as a result of discretization only for small Reynolds numbers **Re**<<1. The solution of such equations for **Re**<<1 is achieved using Newtonian or pseudo-Newtonian algorithms, as well as the pseudo- nonstationary algorithm.

- For large Reynolds numbers **Re>>1**, the Jacobi matrix J becomes ill-conditioned, causing Newton's method to diverge. For this reason, the pseudo-nonstationary algorithm or other approaches like multigrid methods [20, 57] are mostly used in such cases.

Traditionally, Newton's method is used mostly in tandem with the finite-volume method whereas the pseudo-nonstationary method with splitting is more associated with the finite-difference method. Since both methods are related, it is preferable to choose the method that is most compatible with the given problem.

3.7 Exercises on elliptic PDEs

1. Give examples of when an internal point in the 5-point "cross" molecule belongs to a set of the boundary points of the 9-point "compact molecule" routine.

2. Using the modified equation[#], compute an approximation discrepancy of the Laplace equation for the 5-point "cross" routine (Fig. 2.5a).

3. Develop a finite-difference approximation for the equation

   $$u_{xx} + u_{xy} + u_{yy} = 0$$

   using the 9-point "compact molecule" routine (Fig. 2.5b).

4. Using the modified equation[#] compute the approximation discrepancy of the equation in problem 3 using the 9-point "compact molecule" routine (Fig. 2.5b).

5. Develop a finite-difference approximation for the equation

   $$u_{xx} + u_{yy} = x + y$$

This procedure is known as finding a differential approximation for the finite-difference scheme (or **modified equation**) and will be considered in more detail in 5.2.2, so that only a short explanation is given here. The best approximation for the proper mesh is determined by a Taylor series expansion in the peripheral mesh nodes at the point of the mesh central node (i,j), with a further substitution of the expressions obtained into the finite-difference approximation of the original equation. For example, the Laplace equation has the following form on the five-point mesh

$$u_{xx} + u_{yy} = -\frac{1}{12}\left(u_{xxxx}(\Delta x)^2 + u_{yyyy}(\Delta y)^2\right) + \dots ,$$

which is of accuracy $\approx O((\Delta x)^2, (\Delta y)^2)$. See also sections 2.2 and 4.1.1.

and, using the modified equation, compute the approximation discrepancy for the "extended cross" routine (Fig. 2.5c).

6. Develop a finite-difference approximation for the biharmonic equation $\Delta^2 u = 0$, where Δ is the Laplace operator, using the 13-point routine (Fig. 2.5d).

7. A plate, enclosing one side of the square 1m x 1m domain shown in Fig. E-3.1, moves in its own plane with a velocity of 1 m/s, thereby inducing the plane flow of the highly viscous liquid within the domain. Generate a 10x10 grid and use the iterative Jacobi method to compute the streamlines and vorticity lines.

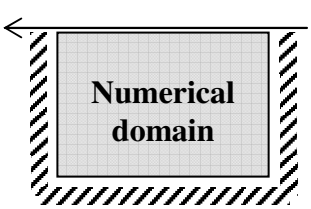


Fig. E-3.1.

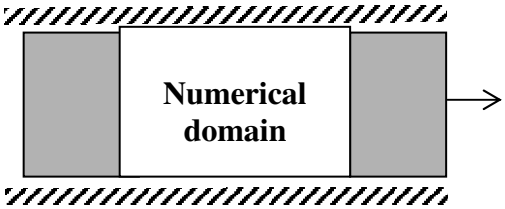


Fig. E-3.2.

It assumed that the process is described by the biharmonic equation $\Delta^2 \psi = 0$, where Δ is the Laplace operator and ψ is the stream function.

8. Capillary blood flow can be modelled by an infinite chain of plane quadratic cylinders of zero buoyancy moving with velocity of 0.0001 m/s in a water-filled gap of width 0.00001 m (Fig. E-3.2.). Generate a 10x10 grid and, using the iterative Gauss-Seidel method with successive overrelaxation, compute the streamlines and vorticity lines. The process is described by the biharmonic equation $\Delta^2 \psi = 0$, where Δ is the Laplace operator, ψ is the stream function. Assume that the blood cells cover the entire capillary and that the distance between them is half the gap width.

9. Fig. E-3.3 shows a solar collector consisting of a plane plate element:

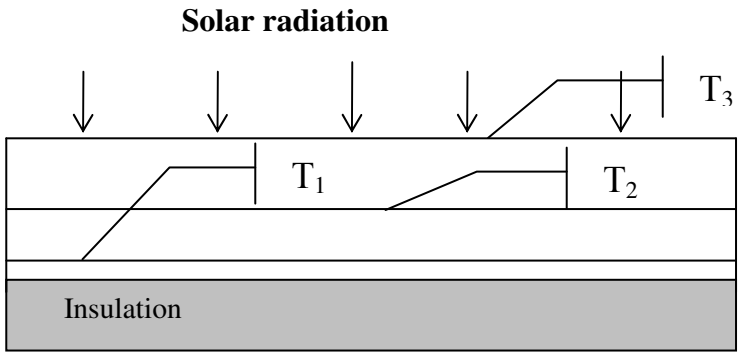


Fig. E-3.

An analysis of its operation gives the following energy balance for the absorber and two glass covers:

$$R_1 = \left(T_1^4 + 0.06823T_1\right) - \left(T_2^4 + 0.05848T_2\right) - 0.01509 = 0\,,$$

$$R_2 = \left(T_1^4 + 0.05848T_1\right) - \left(2T_2^4 + 0.11696T_2\right) + \left(T_3^4 + 0.5848T_3\right) = 0\,, \qquad \text{(E-3.1)}$$

$$R_3 = \left(T_1^4 + 0.05848T_2\right) - \left(2.05T_3^4 + 0.2534T_3\right) + 0.06698 = 0\ .$$

Solve the non-linear equation array (E-3.1) for the absorber temperature T_1 and the temperatures of the two glass covers, T_2 and T_3, $T_i = T_i^o\mathrm{K}/1000$ by applying Newton's method. Compare the computation time for Newton's method with the time required using the partial "freezing" of factorization.

10. For problem 9, use the quasi-Newtonian method with

$$H_{i,i} = 1/J_{i,i}\ ;\ H_{i,j} = 0\ ,\ i \neq j\ .$$

Compare the total computation time with that required for Newton's method.

11. Using the modified equation, compute an approximation discrepancy for the equation in problem 5 using the 9-point “compact molecule” routine (Fig. 2.5b).

12. Using the modified equation, compute an approximation discrepancy for the equation in problem 3 using the 9-point “extended cross” routine (Fig. 2.5c).

13. Develop a finite-difference approximation for the Poisson equation in problem 5 using the 13-point routine (Fig. 2.5d).

14. Using the modified equation, compute an approximation discrepancy for the Laplace equation

$$u_{xx} + u_{yy} = 0$$

using the 9-point “compact molecule” routine (Fig. 2.5b).

4. Basic aspects of the discretization of linear parabolic PDEs

Parabolic second-order partial differential equations describe heat conduction, diffusion processes and flows in boundary layers approach when variables change more rapidly in one direction (across a thin layer) than in others. Examples of boundary-layer flows are: flow in a thin layer attached to a circumfluent body, jet and film flows, flows in capillary channels, etc. Whereas in processes described by elliptic PDEs (e.g. (1.5), (2.27), (3.61)) all perturbations spread into the entire numerical domain, for parabolic PDEs (e.g. (1.6), (3.63), (4.1)) they spread only downstream and not against the flow direction. This is the main physical and numerical difference between elliptic and parabolic PDEs that should be clearly understood before beginning to solve a given problem. This primary feature predetermines the methods that are developed and applied for the solution of boundary-value problems, both in continuum mechanics and in other fields.

4.1 One-dimensional diffusion equation

As regards the development of numerical methods for parabolic PDEs, the linear diffusion equation has the same dissipation mechanism as the equation for a stationary laminar or turbulent gas or fluid flow in a boundary layer, or the non-stationary Navier-Stokes or Reynolds equations. On the other hand, the boundary-layer equations, as well as the non-stationary Navier-Stokes and Burgers' equations, are non-linear, and we consider these problems further, after a detailed analysis of a numerical simulation for non-linear transfer process.

Let us first consider the linear heat conduction equation

$$\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2} = 0 \tag{4.1}$$

as an example of a parabolic PDE. Here the function T can be velocity, temperature, concentration, etc. depending on the problem being considered: diffusion of momentum, heat or mass, respectively. If T is the temperature, then (4.1) describes the heat flux in a rod whose horizontal surfaces are insulated, but which releases heat from its ends to the surroundings (at points A and B in Fig. 4.1).

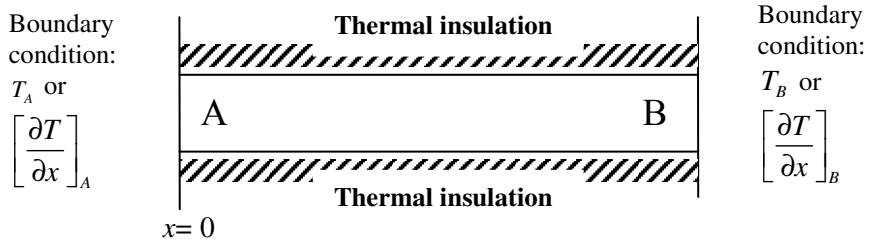


Fig. 4.1. One-dimensional non-stationary heat conduction process.

In general, the boundary and initial conditions should be prescribed. The boundary conditions are prescribed for space arguments (here x), so that the dependent variable is considered as a known function of time. Here, the possibilities are:

- Dirichlet boundary conditions:

$$T(0,t)=\varphi_0(t)\ ;\ T(1,t)=\varphi_1(t)\ ; \tag{4.2}$$

- Neumann boundary conditions:

$$\left[\frac{\partial T}{\partial x}\right]_{x=0}=\psi_0(t)\ ;\ \left[\frac{\partial T}{\partial x}\right]_{x=1}=\psi_1(t)\ ; \tag{4.3}$$

- Robin boundary conditions:

$$\left[\alpha_0(t)\frac{\partial T}{\partial x}+\beta_0(t)T\right]_{x=0}=\chi_0(t);$$
$$\left[\alpha_1(t)\frac{\partial T}{\partial x}+\beta_1(t)T\right]_{x=1}=\chi_1(t)\ . \tag{4.4}$$

In order to get a unique solution for a non-stationary problem, an initial condition should also be stated (Cauchy problem):

$$T(x,0)=\Theta(x). \tag{4.5}$$

For the one-dimensional diffusion equation, we consider both the Cauchy problem and the mixed boundary-value problem.

4.1.1 Explicit approaches for the Cauchy problem

The Cauchy problem, (4.1) and (4.5), consists in the calculation of $T(x,t)$: $\forall t>0,\ -\infty<x<+\infty$ satisfying the equation (4.1) and the initial condition (4.5) on the line $t=0$ (corresponding to the starting time), where $\Theta(x)$ is some known function in the domain: $\forall x\in(-\infty,+\infty)$. To get an approximate solution, the equation (4.1) and the initial condition (4.5) are discretized. The grid generation here is simple: $x_j=j\Delta x,\ t_n=n\Delta t\ \ (j=0,\pm1,...;\ n=0,1,2,...)$, where Δx is the grid step size in Ox (x-coordinate), and Δt is the grid step size in Ot (time). The nodes (x_j,t_n) are considered as internal nodes of the numerical domain if $n\geq1$. For $n=0\ (t=0)$, the nodes are considered to belong to the boundary of the numerical domain. For this grid, the initial condition (4.5) is the following

$$T_{j0}=\theta_j,\ j=0,\pm1,\pm2,... \tag{4.6}$$

where $\theta_j=\Theta(x_j)$.

The discretization of the one-dimensional non-stationary parabolic PDE (4.1) depends on the routine chosen. Using explicit routines (see Fig. 4.2), only one unknown, e.g. T_j^{n+1}, is present on the left-hand side of the algebraic formula obtained by discretization.

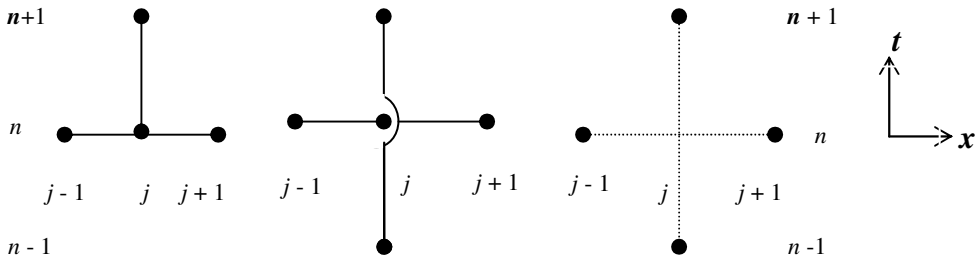


Fig. 4.2. Explicit routines for the diffusion equation

If the time derivative is approximated by a two-point upwind finite-difference scheme and the space derivative is approximated by two-point central differences (FTCS routine – forward in time and central in space), differential equation (4.1) is transformed to

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} - \frac{\alpha\left(T_{j-1}^n - 2T_j^n + T_{j+1}^n\right)}{(\Delta x)^2} = 0 \ . \tag{4.7}$$

In equation (4.7), the space derivative is approximated at the (known) n-th layer in time. After some manipulation, finite-difference approximation (4.7) gives the following algorithm, which can be used for numerical simulation:

$$T_j^{n+1} = sT_{j-1}^n + (1-2s)T_j^n + sT_{j+1}^n, \qquad s = \frac{\alpha\Delta t}{(\Delta x)^2}. \tag{4.8}$$

The main characteristics of any finite-difference scheme (FDS) are its approximation accuracy and the stability conditions. The accuracy of the finite-difference approximation (FDA) on the routine FTCS is obtained by substituting the exact solution T into the finite-difference equation (4.7). For this purpose, every term in equation (4.7) is expanded in a Taylor series about the node (j, n) and then similar terms are simplified[#] to give

[#] This procedure is called the derivation of a differential approximation for the finite-difference scheme (DAFDS) (or modified equation) and is considered in more detail in chapter 5.

$$\left[\frac{\partial T}{\partial t} - \alpha \frac{\partial^2 T}{\partial x^2}\right]_j^n + \underline{\left[\alpha \frac{(\Delta x)^2}{2}\left(s - \frac{1}{6}\right)\frac{\partial^4 T}{\partial x^4}\right]_j^n} + O\left((\Delta t)^2,\ (\Delta x)^4\right) = 0\,. \qquad (4.9)$$

In equation (4.9), the main term of the FDA inaccuracy G_j^n (called truncation error, or T.E.) for equation (4.1) by the FTCS routine is underlined. Thus, according to (4.9), the inaccuracy for the FTCS approximation is $\approx O\left(\Delta t, (\Delta x)^2\right)$. Note that in the case considered, the inaccuracy can be calculated more easily by estimating the approximation scheme for FTCS based on the inaccuracy estimate for the separate derivatives in expressions (2.7)-(2.10) when replacing the differential operators by finite differences. Alternatively, it can be done using the procedure of replacing a differential equation by an FDE through the substitution of the whole operator, as considered in [37]. From another point of view, the differential approximation of a finite-difference scheme allows us to predict some important features of its behaviour. For example, for $s = 1/6$ the terms underlined in (4.9) become zero and the inaccuracy of the approximation is $\approx O\left((\Delta t)^2, (\Delta x)^4\right)$. Stability analysis of the FTCS scheme by von Neumann shows the conditional stability of the scheme for $s \le 1/2$.

In the construction of the algorithm for the FTCS scheme, the approximation of the time derivative is one-sided, and is therefore first-order accurate. Evidently, using the central-difference approximation of the time derivative for the equation

$$\frac{T_j^{n+1} - T_j^{n-1}}{2\Delta t} - \frac{\alpha\left(T_{j-1}^n - 2T_j^n + T_{j+1}^n\right)}{(\Delta x)^2} = 0\,, \qquad (4.10)$$

could increase the overall accuracy. But the Richardson scheme (Fig. 4.2) based on this routine appears to be unconditionally unstable by the von Neumann stability analysis. Thus, the Richardson routine is of no practical use, although it may be modified to give a stable algorithm if T_j^n in equation (4.10) is replaced by $\left(T_j^{n-1} + T_j^{n+1}\right)/2$.

This results in an FDE of the form:

$$\frac{T_j^{n+1} - T_j^{n-1}}{2\Delta t} - \frac{\alpha \left(T_{j-1}^n - \left(T_j^{n-1} + T_j^{n+1}\right) + T_{j+1}^n\right)}{(\Delta x)^2} = 0. \tag{4.11}$$

Equation (4.11), known as the DuFort-Frankel scheme (see Fig. 4.2) can be transformed to the following explicit algorithm:

$$T_j^{n+1} = \left(\frac{2s}{1+2s}\right)\left(T_{j-1}^n + T_{j+1}^n\right) + \left(\frac{1-2s}{1+2s}\right)T_j^{n-1} \quad . \tag{4.12}$$

The von Neumann stability analysis of (4.12) shows the scheme to be stable for arbitrary $s>0$ (absolute stability). However, analysis using DAFDS,

$$\left[\frac{\partial T}{\partial t} - \alpha\frac{\partial^2 T}{\partial x^2} + \alpha\left(\frac{\Delta t}{\Delta x}\right)^2 \frac{\partial^2 T}{\partial t^2}\right]_j^n + O\left((\Delta t)^2,\ (\Delta x)^2\right) = 0, \tag{4.13}$$

results in an additional condition on the method's stability $\left(\Delta t << \Delta x\right)$, which follows from the necessity of the condition that $\Delta t/\Delta x \to 0$ as $\Delta t \to 0$, $\Delta x \to 0$ simultaneously.

Remarks:

- *The DuFort-Frankel scheme is three-layered in all cases except $s = 1/2$, in which case it transforms into the FTCS scheme. The use of a three-layer scheme requires the storage of data for two time layers. Therefore, an alternative two-layer scheme is required for the first time-step.*

- *$s = 1/\sqrt{12}$ optimizes the convergence of the DuFort-Frankel scheme, since the main term of the approximation inaccuracy, $G_j^n = \alpha(\Delta x)^2\left(s^2 - 1/12\right)\partial^4 T/\partial x^4$, which grows as $O\left((\Delta x)^4\right)$ irrespective of the time-step, is then zero.*

- *From a practical point of view, the DuFort-Frankel scheme is substantially restricted by the magnitude of the discrete time-step Δt, despite the fact that this restriction is to do with an accuracy requirement and not a stability requirement, as for the FTCS scheme.*

4.1.2 Implicit approaches for the Cauchy problem

By using implicit schemes for the solution of the Cauchy problem, (4.1) and (4.5), the space term in equation (4.1) is approximated at least partially at the unknown $(n+1)$-st time layer. In practice, this causes the coupling of the equations for every node at that layer and, consequently, requires solving an LAEA at each time-step.

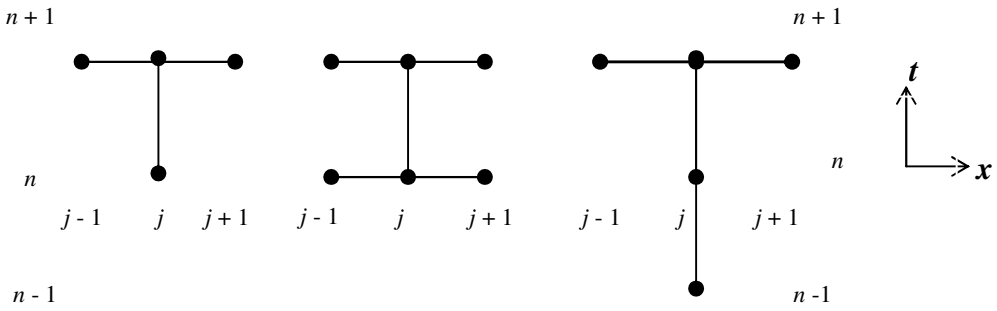
If we apply the two-point backward differencing approximation for a time derivative and two-point central differencing approximation for a space derivative at an unknown layer, the differential equation (4.1) transforms to the following:

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} - \frac{\alpha \left(T_{j-1}^{n+1} - 2T_j^{n+1} + T_{j+1}^{n+1} \right)}{(\Delta x)^2} = 0 \ . \tag{4.14}$$

The fully implicit (FI) algorithm (see Fig. 4.3) has the form

$$-sT_{j-1}^{n+1} + (1+2s)T_j^{n+1} - sT_{j+1}^{n+1} = T_j^n , \tag{4.15}$$

where $s = \alpha \Delta t / (\Delta x)^2$.



FI (fully implicit) Crank-Nicholson G3L (generalized 3-layer)

Fig. 4.3. Implicit routines for the diffusion equation.

The main deficiency term (or T.E.) of the differential approximation for the finite-difference scheme, G_j^n, for the FI algorithm is the following:

$$G_j^n = -\frac{\Delta t}{2}\left(1 + \frac{1}{6s}\right)\left[\frac{\partial^2 T}{\partial t^2}\right]_j^n + O\left((\Delta t)^2,\ (\Delta x)^4\right). \tag{4.16}$$

Thus, the expression obtained in (4.16) for the approximation deficiency is of the same order as for the implicit FTCS scheme for $s \neq 1/6$, although the constant here is somewhat larger. The von Neumann stability analysis shows that the FI scheme is absolutely stable, which makes it evidently preferable to the conditionally stable explicit schemes.

As mentioned earlier, advancement to the next time-step for FI requires the solution of the LAEA (4.15) by the Thomas method for a tridiagonal band matrix. In practice, this requires twice as much computation time as the solution of the same problem by the explicit FTCS scheme. However, the time-step for the implicit scheme may be substantially bigger, as it is restricted only by the accuracy required.

An alternative implicit algorithm for the solution of the diffusion equation (3.1) is the Crank-Nicholson scheme (Fig. 4.3), given by

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} - \alpha\left(0.5 L_{xx} T_j^n + 0.5 L_{xx} T_j^{n+1}\right) = 0, \tag{4.17}$$

where the finite-difference operator is

$$L_{xx} T = \frac{T_{j-1} - 2T_j + T_{j+1}}{(\Delta x)^2}. \tag{4.18}$$

In fact, the Crank-Nicholson scheme approximates a space derivative at an intermediate $(n+1/2)$-th layer in time, thereby raising the accuracy of the time derivative by one order. This is because a differential approximation for the Crank-Nicholson FDS results in an approximation inaccuracy $\approx O\left((\Delta t)^2, (\Delta x)^2\right)$, which is much better than for a fully implicit scheme.

The von Neumann stability analysis indicates the absolute stability of the Crank-Nicholson scheme. FDE (4.17) is easily transformed to the algorithm

$$-0.5sT_{j-1}^{n+1} + (1+s)T_j^{n+1} - 0.5sT_{j+1}^{n+1} = -0.5sT_{j-1}^{n} + (1-s)T_j^{n} - 0.5sT_{j+1}^{n}, \tag{4.19}$$

where $j = \overline{2, N-1}$. The system of FDEs (4.19) is solved by the Thomas method.

Remark: *The Crank-Nicholson scheme is at the edge of the absolutely stable region, and its performance is often unacceptable due to slow convergence and strong oscillations in stiff and pseudo-nonstationary problems.*

The diffusion equation is normally solved by a generalized 3-layer scheme (G3LS), which is obtained from Crank-Nicholson scheme, according to

$$\frac{(1+\gamma)\Delta T_j^{n+1}}{\Delta t} - \frac{\gamma \Delta T_j^{n}}{\Delta t} - \alpha\left[(1-\beta)L_{xx}T_j^{n} + \beta L_{xx}T_j^{n+1}\right] = 0, \tag{4.20}$$

where

$$\Delta T_j^{n} = T_j^{n} - T_j^{n-1},\ L_{xx}T_j^{n} = \frac{T_{j-1}^{n} - 2T_j^{n} + T_{j+1}^{n}}{(\Delta x)^2}. \tag{4.21}$$

The extra time layer in the numerical scheme (Fig. 4.3) requires additional computer memory to store the solution. A more complex algorithm also requires additional execution time. Choosing $\gamma = 0.5$, $\beta = 1.0$, one can get an approximation inaccuracy

for (4.20) of approximately $O\left((\Delta t)^2,(\Delta x)^2\right)$, which is no worse than that in the Crank-Nicholson scheme. But G3LS has no restrictions as regards stiff problems, which is not the case for the Crank-Nicholson scheme. Thus the generalized three-layer scheme (G3LS) finds much broader application, which is its main advantage; in addition, the von Neumann stability analysis shows that it is absolutely stable.

4.1.3 A hybrid scheme for the one-dimensional diffusion equation

The mixed boundary-value problem for PDE (4.1) consists of finding the solution $T(x,t):\ \forall t\in D=\{0\le x\le 1,\ t>0\}$, satisfying the initial condition

$$T(x,0)=\Theta(x) \qquad \text{for } 0\le x\le 1, \tag{4.22}$$

and the boundary conditions

$$\begin{aligned}&\left[\alpha_0(t)\frac{\partial T}{\partial x}+\beta_0(t)T\right]_{x=0}=\chi_0(t)\ ;\\ &\left[\alpha_1(t)\frac{\partial T}{\partial x}+\beta_1(t)T\right]_{x=1}=\chi_1(t)\ .\end{aligned} \tag{4.23}$$

It is supposed that for $\forall t\ \ \alpha_i^2(t)+\beta_i^2(t)>0$, $i=0,1$, and that the conditions (4.22) and (4.23) are compatible at the corner points $(0,0)$ and $(1,0)$.

Now generate the grid: $x_j=j\Delta x;\ t_n=n\Delta t$, $j=\overline{1,N};\ n=0,1,2,\ldots$. If $t\in[0,\pi]$, then the following space and time steps are chosen, respectively: $\Delta x=\frac{1}{N}$, $\Delta t=\frac{\pi}{M}$. Consider the following sets of internal and boundary points, respectively:

$$D_h = \left\{ \left(x_j, t_n \right) : 2 \le j \le N-1,\ n \ge 1 \right\},$$

$$\Gamma_h = \left\{ \left(x_j, t_n \right) : n = \overline{0, M} \text{ for } j = 1; N \text{ and } 1 \le j \le N \text{ for } n = 0 \right\}.$$

Discretization of equation (4.1) is performed in a similar way to that for the Cauchy problem. In solving the Cauchy problem, the only source of error was due to the discretization procedure for PDE (4.1). For the mixed boundary-value problem, an additional obstacle is the necessity to discretize the boundary and initial conditions. One-sided differencing using just the internal domain, i.e.

$$\frac{T_2^{n+1} - T_1^{n+1}}{\Delta x} + O(\Delta x) = \psi_0^{n+1}, \tag{4.24}$$

approximates the space derivative in the Neumann boundary condition so that the space approximation inaccuracy exceeds even the one in the FTCS scheme. This is unacceptable because the lower approximation accuracy of the boundary condition leads to a reduced overall accuracy for the solution.

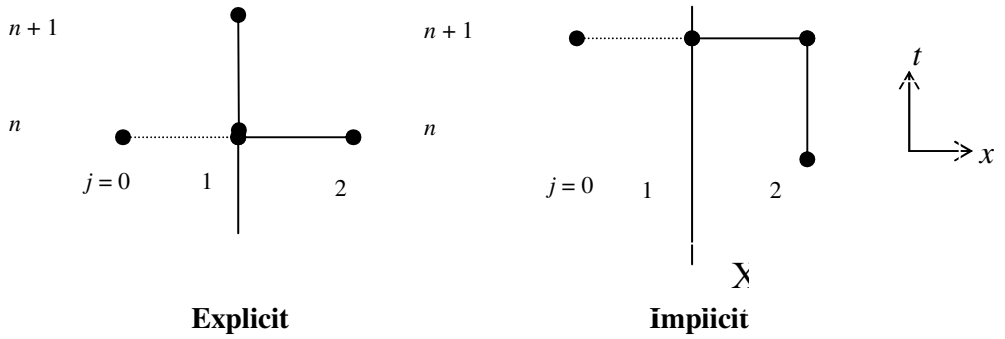


Fig. 4.4. Explicit and implicit Neumann boundary conditions.

The two variants of the Neumann boundary condition proposed in Fig. 4.4 give an approximation inaccuracy of $O\left((\Delta x)^2\right)$. The explicit scheme uses the fictitious node (0,*n*) outside the numerical domain. Application of central space differencing results in

$$\left[\frac{\partial T}{\partial x}\right]_{x=0}^{n} = \psi_0\left(t_n\right) \rightarrow \frac{T_2^n - T_0^n}{2\Delta x} = \psi_0^n . \tag{4.25}$$

Further, the numerical domain can be extended to include the point (0,*n*). Combining equation (4.25) with FDE (4.8) for the point (1, *n*) and eliminating T_0^n, we obtain

$$T_1^{n+1} = -2s\Delta x\psi_0^n + \left(1-2s\right)T_1^n + 2sT_2^n . \tag{4.26}$$

Moreover, the approximation accuracy is of the same order, i.e. $\approx O\left(\Delta t,(\Delta x)^2\right)$, in the whole numerical domain and is the same as for the FTCS scheme. Thus, the numerical scheme is first-order accurate in time and second-order accurate in space.

If the implicit scheme is used instead of (4.25), so that

$$\left[\frac{\partial T}{\partial x}\right]_{x=0}^{n+1} = \psi_0\left(t_{n+1}\right) \rightarrow \frac{T_2^{n+1} - T_0^{n+1}}{2\Delta x} = \psi_0^{n+1} , \tag{4.27}$$

then, coupled with the algorithm for the fully implicit scheme, it becomes

$$\left(1+2s\right)T_1^{n+1} - 2sT_2^{n+1} = T_1^n - 2s\Delta x\psi_0^{n+1}, \tag{4.28}$$

which can be considered as an additional first equation of a block tridiagonal system obtained by discretizing the diffusion equation by the fully implicit (FI) scheme (Fig. 4.3). This extended system is also solved using the Thomas algorithm. The same approximation is used for the Neumann boundary condition both at $x=0$ and $x=1$.

It is to be expected that the application of the additional relations on the boundaries will yield a change in the stability conditions, because the von Neumann stability analysis is applicable only for the internal points of the numerical domain. In our case, the eigenvalues should be calculated directly for the extended matrix of the linear algebraic equation array (LAEA). The eigenvalues must be strictly less than one. The matrix analysis performed by Mitchell and Griffiths [20] did not show any changes in the solution behaviour obtained for the extended LAEA matrices mentioned above. The interaction between the different finite-difference approximation schemes for the equations and boundary conditions will be the topic for separate investigations.

Remarks:

- *The Crank-Nicholson scheme with the Dirichlet boundary conditions is absolutely stable. Using this scheme for the Neumann (Robin) boundary conditions, it is necessary for stability to introduce some additional restrictions on the parameters, because the eigenvalues of an extended LAEA are not all strictly < 1.*
- *Three-layer schemes use, at the first step, a two-layer scheme of the same or higher-order accuracy. If it is not possible to use any scheme of a high enough accuracy at the first step, then Richardson extrapolation, coupled with the Runge rule, can be beneficial.*

4.2 Boundary-value problems for the multi-dimensional diffusion equation (MDDE)

All the numerical schemes considered can be generalized from two to three space variables; therefore a two-dimensional diffusion equation is considered here just for simplicity:

$$\frac{\partial T}{\partial t} - \alpha_x \frac{\partial^2 T}{\partial x^2} - \alpha_y \frac{\partial^2 T}{\partial y^2} = 0. \tag{4.29}$$

The Dirichlet boundary conditions for the domain shown in Fig. 4.5 are:

$$T(0,y,t)=a(y,t), \qquad T(1,y,t)=b(y,t);$$
$$T(x,0,t)=c(y,t), \qquad T(x,1,t)=d(y,t). \tag{4.30}$$

The initial condition is written in the following form:

$$T(x,y,0)=T_0(x,y)\ , \tag{4.31}$$

where $T_0(x,y)$ is a known function of x and y.

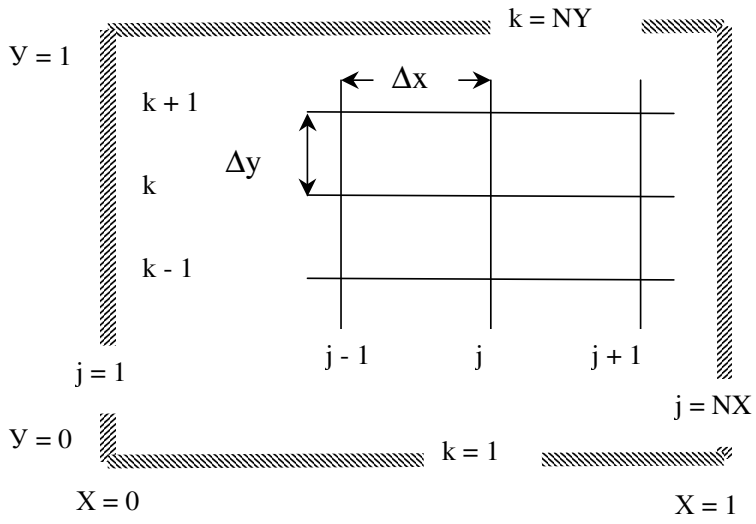


Fig. 4.5. Numerical domain for the two-dimensional diffusion equation.

4.2.1. Explicit and implicit approaches for MDDE

First of all, the typical explicit and implicit schemes considered for the one-dimensional diffusion equation are generalized for the case of two and three variables to see directly their applicability for solving the multi-dimensional diffusion equation.

The FTCS numerical scheme (forward in time, central in space) for a two-dimensional case has the algorithm

$$T_{j,k}^{n+1} = s_x T_{j-1,k}^n + \left(1 - 2s_x - 2s_y\right) T_{j,k}^n + s_x T_{j+1,k}^n + s_y T_{j,k-1}^n + s_y T_{j,k+1}^n, \tag{4.32}$$

where $s_x = \dfrac{\alpha_x \Delta t}{(\Delta x)^2}$, $s_y = \dfrac{\alpha_y \Delta t}{(\Delta y)^2}$ and the discretization error is $\approx O\left(\Delta t, (\Delta x)^2, (\Delta y)^2\right)$; thus, the scheme is first-order in time and second-order in space. The scheme is stable for $s_x + s_y \le 0.5$. Note that the stability condition is twice as strong as the one for the one-dimensional case. This is because, for $s = s_y = s_x$ the stability condition for (4.32) is $s \le 0.25$. If $\alpha = \alpha_x = \alpha_y$ and $\Delta x = \Delta y$, we have the following two-step generalization of the FTCS:

$$T_{j,k}^* = \left(1 + \alpha \Delta t L_{yy}\right) T_{j,k}^n, \; T_{j,k}^* = \left(1 + \alpha \Delta t L_{yy}\right) T_{j,k}^n, \tag{4.33}$$

which is stable for $0 < s \le 0.5$. The space operator has the following form:

$$L_{xx} T_{j,k}^n = \frac{T_{j-1,k}^n - 2T_{j,k}^n + T_{j+1,k}^n}{(\Delta x)^2} \; . \tag{4.34}$$

Remark: *it is better to use the three-dimensional FTCS for* $\alpha = \alpha_x = \alpha_y = \alpha_z$ *and* $\Delta x = \Delta y = \Delta z$ *in three steps; this keeps it stable for* $0 < s \le 1/6$.

The "classic" scheme is more interesting than FTCS for the two-dimensional diffusion equation, which can be considered as a two-step FTCS:

1. First, algorithm (4.32) is applied to all nodes with even $\sum\left(j + k + n\right)$;
2. Then, for the nodes with odd $\sum\left(j + k + n\right)$, the following equation is solved:

$$\left(1 + 2s_x + 2s_y\right) \; T_{j,k}^{n+1} = T_{j,k}^n + s_x \left(T_{j-1,k}^{n+1} + T_{j+1,k}^{n+1}\right) + s_y \left(T_{j,k-1}^{n+1} + T_{j,k+1}^{n+1}\right), \tag{4.35}$$

where the terms on the right-hand side of equation (4.35), calculated at time t_{n+1}, are known from step 1. The method is simple in practice and gives accuracy $\approx O\left(\Delta t,(\Delta x)^2,(\Delta y)^2\right)$, as well as being unconditionally (absolutely) stable.

Now, the fully implicit scheme can be built in similar fashion to the one-dimensional problem considered above. Consequently, the algorithm is as follows:

$$-s_x T_{j-1,k}^{n+1}+\left(1+2s_x+2s_y\right)\ T_{j,k}^{n+1}-s_x T_{j+1,k}^{n+1}-s_y T_{j,k-1}^{n+1}-s_y T_{j,k+1}^{n+1}=T_{j,k}^{n}. \qquad (4.36)$$

This absolutely stable scheme has discretization error $\approx O\left(\Delta t,(\Delta x)^2,(\Delta y)^2\right)$. The equations of the system (4.36) can be numbered in such a way as to have the three first terms constitute the three diagonals, with the remaining two terms being placed at a distance of *NX*-2 nodes from the first ones (see Fig. 4.5). From the latter, it follows that the Thomas algorithm cannot be used. An application of the Gauss method (or even some special methods available for sparse matrices) for the solution of the equation array (4.36) at the (*n*+1)-st layer would be very uneconomical, and therefore implicit algorithms are not used in practice for multi-dimensional problems directly.

4.2.2 The alternating-direction-implicit (ADI) technique

The splitting of the solution algorithm into two half steps, which together give one whole time step, counteract the problems mentioned above. At every half step, only the terms in a corresponding direction are formulated implicitly. There are only three such terms at each half step; they are clustered around the main diagonal, so that at each half time step, the effective Thomas algorithm is applicable.

The best-known split method is the classic alternating-direction-implicit (ADI) procedure developed in the mid-1950s by Peaceman, Rackford, and Douglas [16, 62].

Later on, the method was developed and demonstrated by Janenko [35] for different and more complex equations. The ADI scheme for equation (4.29) is written in the form of two half time steps. At the first half step, the solution T is known at the layer t_n, but unknown at $t_{n+\frac{1}{2}}$ (marked *), so that the equations are as follows:

$$\begin{cases} \dfrac{T^*_{j,k}-T^n_{j,k}}{0.5\Delta t}-\alpha_x L_{xx}T^*_{j,k}-\alpha_y L_{yy}T^n_{j,k}=0, \\ \dfrac{T^{n+1}_{j,k}-T^*_{j,k}}{0.5\Delta t}-\alpha_x L_{xx}T^*_{j,k}-\alpha_y L_{yy}T^{n+1}_{j,k}=0. \end{cases} \tag{4.37}$$

Note that the unknowns T^* are coupled only in the direction $\rightarrow Ox$ (with k constant). The first equation of the equation array (4.37) can be presented in the form:

$$\begin{aligned} &-0.5s_x T^*_{j-1,k}+(1+s_x)T^*_{j,k}-0.5s_x T^*_{j+1,k}= \\ &=0.5s_y T^n_{j,k-1}+(1-s_y)T^n_{j,k}+0.5s_y T^n_{j,k+1}. \end{aligned} \tag{4.38}$$

To compute T^*, one needs to solve an equation array with a block tridiagonal matrix structure by the Thomas method.

At the second half step, the other equation of the system (4.37) is considered in the following form:

$$\begin{aligned} &-0.5s_y T^{n+1}_{j,k-1}+(1+s_y)T^{n+1}_{j,k}-0.5s_y T^{n+1}_{j,k+1}= \\ &=0.5s_x T^*_{j-1,k}+(1-s_x)T^*_{j,k}+0.5s_x T^*_{j+1,k}. \end{aligned} \tag{4.39}$$

An equation array with the block tridiagonal matrix (4.39) is solved in the direction Oy (now k is constant) also using the Thomas algorithm, as was previously done for Ox .

The ADI scheme has accuracy $\approx O\left((\Delta t)^2,(\Delta x)^2,(\Delta y)^2\right)$ and is absolutely stable for a complete time step, although it is only conditionally stable for each variable (each half time-step) separately. The approximation of the differential operator is carried out here for a complete time step. Thus, the numerical scheme has second-order accuracy both in time and space.

Remarks:

- *Generalization of the ADI method for four time layers* $\left(n,\ n+1/3,\ n+2/3,\ n+1\right)$ *gives a stable scheme with accuracy* $\approx O\left(\Delta t,(\Delta x)^2,(\Delta y)^2,(\Delta z)^2\right)$. *Here,* $s_x,s_y,s_z \le 1.5$, $s_z=\alpha_z\Delta t/(\Delta z)^2$.

- *To achieve approximation accuracy* $\approx O\left((\Delta t)^2,(\Delta x)^2,(\Delta y)^2\right)$ *for the ADI scheme, one needs to introduce, for the intermediate solution* T^*, *boundary parameters which are compatible with the algorithms (4.38) and (4.39) for the internal points. For example, the Dirichlet boundary condition for* $T^*_{Nx,k}=b_k^{n+1/2}$ *at* $x=1$ *results in accuracy* $O\left(\Delta t\right)$. *To get accuracy of the order* $O\left(\left(\Delta t\right)^2\right)$, *it is necessary to state the following boundary condition:*

$$T^*_{NX,k}=0.5\left(b_k^n+b_k^{n+1}\right)-0.25\Delta t L_{yy}\left(b_k^{n+1}-b_k^n\right). \tag{4.40}$$

4.2.3 The method of fractional steps (MFS)

The method of fractional steps (MFS) was developed by Janenko [35]. The main strategy of the ADI method was first to discretize, and then to modify the algebraic equations in order to construct “one-dimensional” algorithms of the type (4.38)-(4.39). The alternative strategy of MFS lies in the splitting of a basic equation into several equations, each of which is one-dimensional, so that instead of equation (4.29), the following equation array is obtained:

$$0.5\frac{\partial T}{\partial t}-\alpha_y\frac{\partial^2 T}{\partial y^2}=0, \qquad 0.5\frac{\partial T}{\partial t}-\alpha_x\frac{\partial^2 T}{\partial x^2}=0. \tag{4.41}$$

Each of the equations in (4.41) is discretized and solved successively for each time step using the well-known one-dimensional numerical schemes.

The explicit approximation of (4.41) for MFS is the following:

$$T_{j,k}^{n+\frac{1}{2}}=\left(1+\alpha_y\Delta t L_{yy}\right)T_{j,k}^{n}, \qquad T_{j,k}^{n+1}=\left(1+\alpha_x\Delta t L_{xx}\right)T_{j,k}^{n+\frac{1}{2}}. \tag{4.42}$$

If $\alpha_x=\alpha_y=\alpha$, the scheme (4.42) coincides with the two-step FTCS scheme (4.33). The explicit algorithm has accuracy $\approx O\left(\Delta t,(\Delta x)^2,(\Delta y)^2\right)$ and, for $\Delta x=\Delta y$, is stable for $s\le 0.5$.

The implicit method of fractional steps (MFS), when applied to equation array (4.41) using the Crank-Nicholson scheme, yields the following approximation:

$$\begin{aligned}&\left(1-0.5\alpha_y\Delta t L_{yy}\right)T_{j,k}^{n+1/2}=\left(1+0.5\alpha_y\Delta t L_{yy}\right)T_{j,k}^{n},\\ &\left(1-0.5\alpha_x\Delta t L_{xx}\right)T_{j,k}^{n+1}=\left(1+0.5\alpha_x\Delta t L_{xx}\right)T_{j,k}^{n+1/2}.\end{aligned} \tag{4.43}$$

The system of equations (4.43) results in an equation array with a block tridiagonal matrix along the numerical grid lines, which are parallel to the coordinate axes *Ox* and *Oy*, respectively. Thus the solution at each half-step is found using the Thomas algorithm. This scheme has an order of accuracy $\approx O\left((\Delta t)^2,(\Delta x)^2,(\Delta y)^2\right)$ and is absolutely stable for proper boundary conditions, both for two-dimensional as well as three-dimensional cases.

The principal difference between the MFS and ADI methods lies in the fulfilment of the boundary conditions. Using the MFS for the domain in Fig. 4.5, if a Dirichlet condition is stated at the boundary, so that

$$x=1, \qquad T_{NX,k}^{n+\frac{1}{2}} = b_k^{n+\frac{1}{2}}, \tag{4.44}$$

then the implementation of similar conditions at other boundaries result in a decrease of the common accuracy to $\approx O(\Delta t)$. The correct explicit MFS (4.42) at $x=1$ is:

$$x=1, \qquad T_{NX,k}^{n+\frac{1}{2}} = \left(1+\alpha_y \Delta t L_{yy}\right) b_k^n. \tag{4.45}$$

By analogy with an implicit method of fractional steps (MFS), the numerical scheme (4.43) at $x=1$ yields

$$\left(1-0.5\alpha_y \Delta t L_{yy}\right) T_{NX,k}^{n+\frac{1}{2}} = \left(1+0.5\alpha_y \Delta t L_{yy}\right) b_k^n. \tag{4.46}$$

A number of problems in mechanics solved by the MFS are presented in the monograph [57].

Remark: *MFS is not suited for the solution of elliptic equations by the pseudo-nonstationary method, because it does not allow a direct calculation of the discrepancy (Residual head sequence):*

$$RHS = \left(\alpha_x L_{xx} + \alpha_y L_{yy}\right) T_{j,k}^n. \tag{4.47}$$

4.2.4 Example of a simulation of heterogeneous thermal-hydraulic processes in granular media by MFS

A mathematical model for the numerical simulation of a compressible fluid (steam) flow through a volumetrically heated porous bed, with particular consideration of non-thermal local equilibrium effects, is formulated and solved numerically using the method of fractional steps. Fig. 4.6 depicts schematically the problem considered. A two-dimensional self-heated porous packed bed, which consists of K homogeneous layers, is filled with fluid, which moves from the bottom to the top, and is initially at a known temperature distribution.

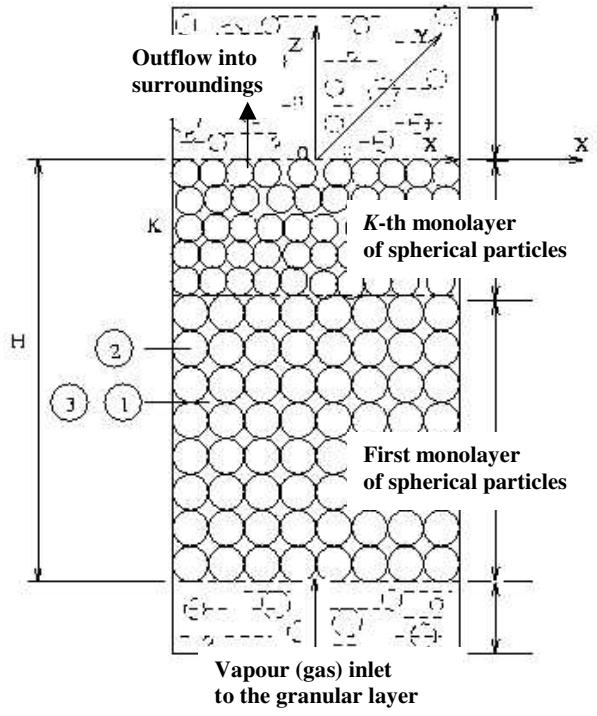


Fig. 4.6. Structural scheme of the heterogeneous “particle-fluid” media

In the development of a mathematical model to study this problem, the following assumptions were employed:

- The flow is single phase and compressible (steam)
- The particle sizes are significantly larger than molecular-kinetic sizes, but significantly smaller than the characteristic size of the system
- Physical properties, such as thermal conductivity, dynamic viscosity, etc. are temperature-dependent
- Solid particles are immovable and the porosity is constant in each monolayer.

The mathematical formulation of the problem is as follows. Based on the equations for a saturated granular layer introduced by Nigmatulin[1] using a heterogeneous approach, the mathematical model for the system, presented schematically in Fig. 4.6, was developed by Kazachkov[2] for the mathematical simulation of non-stationary non-isothermal filtration processes. In a two-dimensional case, the mathematical model for this system can be presented as follows.

Continuity equation for the compressible fluid (vapour or gas) flow:

$$\frac{\partial \rho_1^0}{\partial t} = -\rho_1^0 \left(\frac{\partial u_1}{\partial x} + \frac{\partial w_1}{\partial z} \right), \tag{4.48}$$

Continuity equation for the fluid flow and the layer particles:

$$\rho_1 \frac{\partial \vec{v}}{\partial t} + \nabla p_1 + (\rho_1 + \rho_2)\vec{g} = \nabla \sigma_k , \tag{4.49}$$

[1] R.I. Nigmatulin. *The Basics of Mechanics of Heterogeneous Media*. Moscow: Nauka.- 1978.

[2] I.V. Kazachkov, *On the Mathematical Simulation of Processes of Non-stationary Non-isothermal Filtration in Geothermal Systems//*. Numerical and Applied Mathematics.- Kiev.- Kiev State University.- 1986.- Vol. 60.

$$\rho_1^0(1+0.5\alpha_2)\frac{\partial \vec{v}}{\partial t}+\nabla p_1+\mu_1\frac{\vec{v}}{K_0}\left(\frac{\alpha_1}{\alpha_{10}}\right)^{-n}+\rho_1^0\vec{g}=0\,, \tag{4.50}$$

where ρ_1^0 and ρ_1 are the real vapour (gas) density and partial density in the heterogeneous mixture, respectively, and K_0 is permeability of the layer. The density of the two-phase mixture is $\rho=\alpha_1\rho_1^0+\alpha_2\rho_2$. The other parameters of the two-phase system are calculated in a similar way. The first inertia term in (4.50) accounts for the associated mass for the spherical particle in a non-stationary flow, and the third term expresses the viscous force acting on the particles.

The energy conservation equations are written separately for the two phases of the saturated granular layer (fluid and particles) and the impermeable surrounding medium:

$$\rho_1 c_{v1}\left(\frac{\partial T_1}{\partial t}+\vec{v}.\nabla T_1\right)=\alpha_1 R T_1\frac{\partial \rho_1^0}{\partial t}+\nabla(k_1\nabla T_1)+Q_\Sigma+Q_V+\mu_1(u_1^2+w_1^2)\frac{\alpha_1}{K}\,, \tag{4.51}$$

$$\rho_j c_j\frac{\partial T_j}{\partial t}=\nabla(k_j\nabla T_j)+(j-3)Q_\Sigma\,, \tag{4.52}$$

where $K=K_0(\alpha_1/\alpha_{10})^n$, $\vec{v}=\{u_1,w_1\}$, $k_1=\mu_1 c_{p1}/\mathbf{Pr}$, $\mu_1=\mu_{10}(T_1/T_{10})^m$, j=2,3, and m=0.5-1.0, with the Prandtl number given by $\mathbf{Pr}=\mu_{10}c_{p10}/k_{10}$. The values with "0" indices are taken at the fixed temperature T_{j0}, α is the volume fraction of the corresponding phase, σ, μ, k are the stress tensor, dynamic viscosity and heat conductivity, respectively.

Because we consider a multiphase continuum mechanics problem here, it is supposed from the theory of multiphase dynamics that $\alpha_1+\alpha_2=1$, $\rho=\alpha_1\rho_1+\alpha_2\rho_2$ and so on. Equation (4.52) for j=2 is the energy conservation equation for particles of the permeable layer, and for j=3 it is the equation for the impermeable surroundings. In equations (4.51)

and (4.52), the terms Q_Σ, Q_V are the specific solid-liquid interface heat flux and the volume heat source, respectively.

In the system described above, the vertical pressure gradient is due to gravity ($\vec{g}$) and the temperature gradient, whereas horizontally it is due only to the temperature gradient (thermal convection). Thus, the vertical velocity is greater than the horizontal one. The vertical velocity component is calculated from equations (4.49) and (4.50) as follows:

$$w_1 = (\rho_1 + \rho_2 - \rho_1^0)\frac{gK}{\mu_1},$$

which corresponds to a first-order approximation to Darcy's law since $\rho_1^0 << \rho_1 + \rho_2$. Since the horizontal velocity component u_1 is small in comparison with the vertical, Darcy's law can be also used.

For the system of parabolic second order PDEs (4.48)-(4.52), the following initial and boundary conditions are prescribed:

Initial conditions

$$t = 0, \quad p_1 = p_1^0(x,z), \quad T_j = T_j^0(x,z) \quad , \quad j=1, 2, 3; \tag{4.53}$$

Boundary conditions

The system is considered to be symmetric relative to the vertical axis. The normal velocity is zero at the impermeable boundary, where the temperature and heat flux must be continuous:

$$x = x_L, \qquad u_1 = 0, \qquad k_2\frac{\partial T_2}{\partial x} = k_3\frac{\partial T_3}{\partial x} \quad . \tag{4.54}$$

We expect the temperature in the surrounding impermeable medium to be stable far from the boundary of a permeable layer:

$$x = x_\infty , \quad \frac{\partial T_3}{\partial x} = 0 . \tag{4.55}$$

The temperature of the inlet fluid (at the lower boundary of the porous layer) is assumed to be known:

$$z = -H , \qquad T_j = T_{jH} . \tag{4.56}$$

At the upper boundary (z=0, the outlet of the layer), either a boundary condition similar to (4.56) can be prescribed, or a condition based on heat transfer with the surroundings may be considered.

Thus, the system of parabolic second-order PDEs (4.48)-(4.52) is solved with initial and boundary conditions (4.53)-(4.56). For the numerical solution and further analysis of the results, it is more convenient to consider the boundary-value problem in dimensionless form. For this purpose, the following length, time, velocity, pressure and temperature scales are introduced: H, H^2/a_2^0, a_2^0/H, $\mu_{10}a_2^0/K_0$ and ΔT, where ΔT is the characteristic temperature difference in the system. Now the equation array (4.48)-(4.52) is presented in the following dimensionless form:

$$u_1 = -\frac{\partial p_1}{\partial x}\left(\frac{T_{10}}{T_1}\right)^m , \qquad w_1 = (1-\alpha_1)\left[\mathbf{Pe} - \kappa_\rho \mathbf{Ra}^*(T_2 - T_{20}) - \mathbf{Re}_*^2 \frac{p_1}{T_1}\right]\left(\frac{T_{10}}{T_1}\right)^m ,$$

$$\frac{\partial \rho_1^0}{\partial \mathbf{Fo}} = -\rho_1^0\left(\frac{\partial u_1}{\partial x} + \frac{\partial w_1}{\partial z}\right), \qquad \frac{\partial p_1}{\partial z} = \mathbf{Pe}(\alpha_1 - 1)\left[1 - \Delta_2(T_2 - T_{20})\right] - \mathbf{Re}_*^2 \frac{\alpha_1 p_1}{T_1} ,$$

$$\frac{\partial T_1}{\partial \mathbf{Fo}} = (1-\gamma_1)T_1\left(\frac{\partial u_1}{\partial x} + \frac{\partial w_1}{\partial z}\right) - \left(u_1\frac{\partial T_1}{\partial x} + w_1\frac{\partial T_1}{\partial z}\right) + (\gamma_1 - 1)(u_1^2 + w_1^2)\left(\frac{T_{10}}{T_1}\right)^m \frac{T_1}{p_1} +$$

$$+\frac{\gamma_1 \mathbf{Pe}\left(T_1/T_{10}\right)^m}{\alpha_1 \kappa_a \kappa_\rho \mathbf{Re}_*^2 p_1}\left\{T_1\left(\frac{\partial^2 T_1}{\partial x^2}+\frac{\partial^2 T_1}{\partial z^2}\right)+m\left[\left(\frac{\partial T_1}{\partial x}\right)^2+\left(\frac{\partial T_1}{\partial z}\right)^2\right]+\xi \mathbf{Nu}_1 T_1(T_2-T_1)\right\},$$

$$\frac{\partial T_2}{\partial \mathbf{Fo}}=\frac{(1-\alpha_1)^{-1}}{1-\Delta_2(T_2-T_{20})}\left[\frac{\partial^2 T_2}{\partial x^2}+\frac{\partial^2 T_2}{\partial z^2}+\xi\frac{\mathbf{Nu}_1}{\kappa_k}\left(\frac{T_1}{T_{10}}\right)^m(T_1-T_2)\right], \tag{4.57}$$

$$\frac{\partial T_3}{\partial \mathbf{Fo}}=a_{32}\left(\frac{\partial^2 T_3}{\partial x^2}+\frac{\partial^2 T_3}{\partial z^2}\right).$$

Further, the initial and boundary conditions (4.53)-(4.56) are transformed into the following dimensionless form:

$$\mathbf{Fo}=0,\quad p_1=p_1^0(x,z),\quad T_j=T_j^0(x,z),\quad j=1, 2, 3;$$

$$z=0,\quad T_j=T_{jH};\qquad z=-1,\quad T_j=T_{jH}. \tag{4.58}$$

$$x=x_L,\quad u_1=0,\quad \frac{\partial T_2}{\partial x}=a_{32}\frac{\partial T_3}{\partial x};\qquad x=x_\infty,\quad \frac{\partial T_3}{\partial x}=0.$$

The problem is solved for constant layer porosity, α_1. Here, $\mathbf{Pe}=w_0 H/a_2^0$ is the Péclet number, $w_0=\rho_{20}^0 Kg/\mu_{10}$ is the characteristic filtration velocity, a is the heat diffusivity, e.g. $a_1^0=k_1^0/(c_{p1}\rho_{10}^0)$. The following dimensionless criteria are important for the problem: $\mathbf{Re}_v=w_0 b_1/\nu_{10}$, the Reynolds number for the characteristic pores of the permeable layer; $\mathbf{Re}_*^2=gH/(R\Delta T)$; $\mathbf{Ra}^*=\mathbf{GrPr}^*\mathbf{Da}$, the Rayleigh number; $\mathbf{Gr}$, $\mathbf{Pr}^*$ and $\mathbf{Da}$ - the Grashof, Prandtl and Darcy numbers, respectively, where $\mathbf{Gr}=g\Delta_2 H^3\nu_{10}$, $\mathbf{Da}=K/H^2$; $\mathbf{Nu}_1=2+0.6\mathbf{Pr}^{1/3}\mathbf{Re}_v^{1/2}$, the Nusselt number; $\mathbf{Fo}=a_2^0 t/H$, the Fourier number. Other parameters are as follows: $\kappa_a=a_2^0/a_1^0$; $\kappa_\rho=\rho_{20}^0/\rho_{10}^0$; $\kappa_k=k_2/k_1^0$; $\Delta_2=\Delta T\beta_{T2}$;

$\gamma_1 = c_{p1}/c_{v1}$; $a_{32} = a_3/a_2^0$; $\xi = s_{12}H^2/b_1$ $\xi = s_{12}H^2/b_1$ (a parameter describing the structure of the granular layer); s_{12}, the specific interface area; $b_1 = b\sqrt{2(2-\pi/3)\pi}$, the character pore radius, where b is the particle radius (which is constant in each monolayer).

In the multiphase system considered here, the interactions of three different processes occur: non-thermal equilibrium between fluid and solid particles in the layer, the mutual influence of non-linear processes and non-linearity of the physical properties of fluid and particles (the fluid properties are strongly dependent on temperature and other parameters of the system). The first feature mentioned above is connected with the term $\xi(T_1 - T_2)$, which describes the local heat transfer between the particles and the flow. From the mathematical point of view, it causes some limitation on the parameter ξ because the term $\xi(T_1 - T_2)$ in the energy equations for solid particles and fluid flow appears to be huge for very small particles. These energy equations contain large terms that multiply small ones because, for small particles, the temperature difference $(T_1 - T_2)$ tends to zero. In the limit as the temperature difference tends to zero, one energy equation for the homogeneous mixture should replace these two equations.

Another interesting new phenomenon in the system is due to the localization of dissipative processes caused by a non-linear heat conductivity. This phenomenon was studied first by Samarskii et. al.[3] for quasi-linear parabolic equations, e.g. the one-dimensional heat conduction equation with a non-linear heat conductivity coefficient $k = k_0 T^m$ (m=0.5–1.0). In our case, all these phenomena are interconnected, making the problem very interesting but complex.

[3] A.A.Samarskii, V.A.Galaktionov, S.P. Kurdjumov and A.P.Mikhailov. *Blow-up in Problems for Quasilinear Parabolic Equations.- Moscow: Nauka .- 1987* (in Russian).

Numerical solution of the boundary-value problem (4.57)-(4.58) can be performed using the method of fractional steps [35]. Splitting between the spatial variables transforms the two-dimensional problem into two separate one-dimensional problems.

On the lower half of the time-step (first semi-step):

$$\frac{1}{2}\frac{\partial T_1}{\partial \mathbf{Fo}} = (1-\gamma_1)T_1\frac{\partial u_1}{\partial x} - u_1\frac{\partial T_1}{\partial x} + (\gamma_1 - 1)u_1^2\left(\frac{T_{10}}{T_1}\right)^m \frac{T_1}{p_1} +$$

$$+\frac{\gamma_1 \mathbf{Pe}\left(T_1/T_{10}\right)^m}{\alpha_1 \kappa_a \kappa_\rho \mathbf{Re}_*^2 p_1}\left[T_1\frac{\partial^2 T_1}{\partial x^2} + m\left(\frac{\partial T_1}{\partial x}\right)^2 + \alpha\xi \mathbf{Nu}_1 T_1(T_2 - T_1)\right],$$

$$\frac{1}{2}\frac{\partial T_2}{\partial \mathbf{Fo}} = \frac{(1-\alpha_1)^{-1}}{1-\Delta_2(T_2 - T_{20})}\left[\frac{\partial^2 T_2}{\partial x^2} + \alpha\xi\frac{\mathbf{Nu}_1}{\kappa_k}\left(\frac{T_1}{T_{10}}\right)^m (T_1 - T_2)\right], \tag{4.59}$$

$$\frac{1}{2}\frac{\partial T_3}{\partial \mathbf{Fo}} = a_{32}\frac{\partial^2 T_3}{\partial x^2}, \qquad \frac{1}{2}\frac{\partial \rho_1^0}{\partial \mathbf{Fo}} = -\rho_1^0\frac{\partial u_1}{\partial x}.$$

On the upper half of the time-step (second semi-step):

$$\frac{1}{2}\frac{\partial T_1}{\partial \mathbf{Fo}} = (1-\gamma_1)T_1\frac{\partial w_1}{\partial z} - w_1\frac{\partial T_1}{\partial z} + (\gamma_1 - 1)w_1^2\left(\frac{T_{10}}{T_1}\right)^m \frac{T_1}{p_1} +$$

$$+\frac{\gamma_1 \mathbf{Pe}\left(T_1/T_{10}\right)^m}{\alpha_1 \kappa_a \kappa_\rho \mathbf{Re}_*^2 p_1}\left[T_1\frac{\partial^2 T_1}{\partial z^2} + m\left(\frac{\partial T_1}{\partial z}\right)^2 + (1-\alpha)\xi \mathbf{Nu}_1 T_1(T_2 - T_1)\right],$$

$$\frac{1}{2}\frac{\partial T_2}{\partial \mathbf{Fo}} = \frac{(1-\alpha_1)^{-1}}{1-\Delta_2(T_2 - T_{20})}\left[\frac{\partial^2 T_2}{\partial z^2} + (1-\alpha)\xi\frac{\mathbf{Nu}_1}{\kappa_k}\left(\frac{T_1}{T_{10}}\right)^m (T_1 - T_2)\right], \tag{4.60}$$

$$\frac{1}{2}\frac{\partial T_3}{\partial \mathbf{Fo}} = a_{32}\frac{\partial^2 T_3}{\partial z^2}, \qquad \frac{1}{2}\frac{\partial \rho_1^0}{\partial \mathbf{Fo}} = -\rho_1^0\frac{\partial w_1}{\partial z},$$

where α is an approximation parameter: $\alpha \in [0;1]$. The derivatives inside the two-dimensional numerical domain are approximated by second-order central differences, so that

$$\frac{\partial f}{\partial x} = \frac{f_n^{i+1,j} - f_n^{i-1,j}}{2\Delta x}, \qquad \frac{\partial^2 f}{\partial x^2} = \frac{f_n^{i+1,j} - 2f_n^{i,j} + f_n^{i-1,j}}{(\Delta x)^2},$$

and at the boundary of the numerical domain:

$$\frac{\partial f}{\partial x} = \frac{f_n^{3,j} - 2f_n^{2,j} + f_n^{1,j}}{2\Delta x}.$$

This is also done in a similar way with respect to the coordinate *z*. The time derivatives are calculated at each point (*i*,*j*) of the numerical domain by first-order forward differences, according to

$$\frac{\partial f^{i,j}}{\partial \mathbf{Fo}} = \frac{f_{n+1}^{i,j} - f_n^{i,j}}{\Delta \tau},$$

where $\Delta\tau$ is the time step.

The algorithm for the solution of the equation array (4.59) and (4.60) requires two Thomas marching procedures (for *x* and *z* separately) at each half time step. The full approximation of the PDEs is carried out for the whole time step. The computer code developed using this algorithm is stable and requires less than an hour on a PC to calculate solutions for this complex process for various parameters. Some results of the numerical simulation and validation of the model against experimental data obtained[4] at the Royal Institute of Technology (Stockholm, Nuclear Power Safety Dept.) are presented in Figs. 4.7- 4.10.

[4] I.V. Kazachkov, M.J. Konovalikhin. *A Model of the Steam Flow Through the Volumetrically Heated Particle Bed//* Int. J. of Thermal Sciences.- 2002.- Vol. 8.

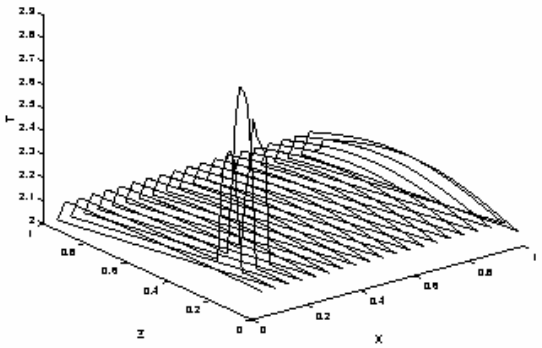


Figure 7: Temperature distribution in the bed (test strat-2.2) Calculations performed at the power density 0.4 MW/m^3.

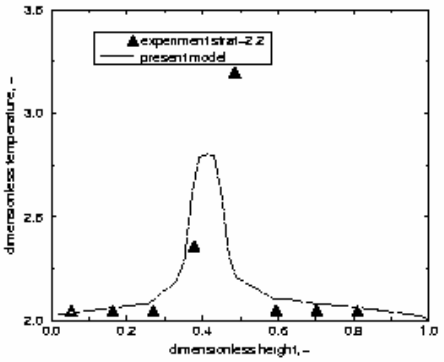


Figure 9: Temperature distribution along Z-axis at X=0.2

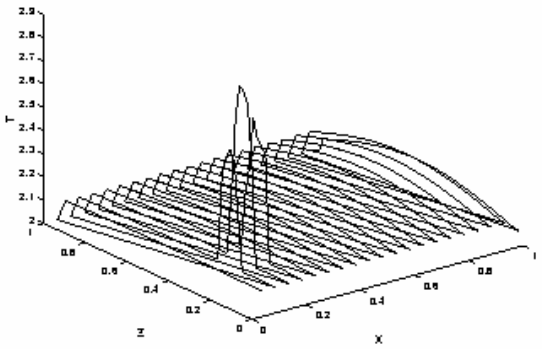


Figure 8: Temperature distribution in the bed (test strat-3.2) Calculations performed at the power density 0.5 MW/m^3.

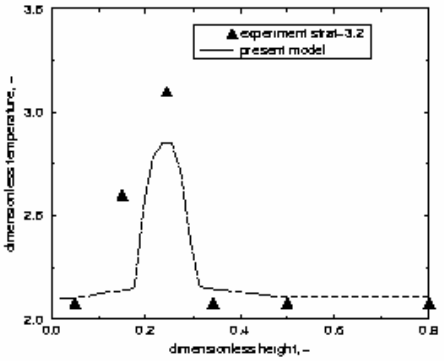


Figure 10: Temperature distribution along Z-axis at X=0.2

Figs. 4.7-4.10. Some results of the simulation of steam flow through a volumetrically heated saturated particle bed and comparison with experimental data.

The results show clearly observed new phenomenon of the localization of dissipative processes caused by non-linear heat conductivity. Local abnormal heating (blow-up) in the particle bed is available by certain above-mentioned conditions.

4.2.5 Generalized splitting schemes for the solution of MDDEs

Now let us consider a generalization of the idea of splitting to find the solution of multi-dimensional diffusion equations. The implicit finite-difference scheme in general form for the two-dimensional diffusion equation (4.29) is

$$\frac{\Delta T_{j,k}^{n+1}}{\Delta t}-(1-\beta)\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^{n}-\beta\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^{n+1}=0, \tag{4.61}$$

where $\Delta T_{j,k}^{n+1}=T_{j,k}^{n+1}-T_{j,k}^{n}$ is considered as a correction to the solution at the n-th step required for the transition to the step (n+1). Therefore, it is useful to compute $\Delta T_{j,k}^{n+1}$ explicitly. The role of the parameter β in equation (4.61) is to provide weighting factors for the terms at the n-th and (n+1)-st time steps.

Using the Taylor series,

$$T_{j,k}^{n+1}=T_{j,k}^{n}+\Delta t\left[\frac{\partial T}{\partial t}\right]_{j,k}^{n}+\frac{1}{2}(\Delta t)^2\left[\frac{\partial^2 T}{\partial t^2}\right]_{j,k}^{n}+\ldots\ ,$$

or, in finite-difference form:

$$T_{j,k}^{n+1}=T_{j,k}^{n}+\Delta t\ \left(\frac{\Delta T_{j,k}^{n}}{\Delta t}\right)+O\left((\Delta t)^2\right), \tag{4.62}$$

the substitution of (4.62) into equation (4.61), after some rearrangement of the terms, yields

$$\left[1-\beta\Delta t\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)\right]\Delta T_{j,k}^{n+1}=\Delta t\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^{n}. \tag{4.63}$$

To apply splitting, replace in equation (4.63) the operator to the left by the product $\left(1-\beta\Delta t\alpha_x L_{xx}\right)\left(1-\beta\Delta t\alpha_y L_{yy}\right)$, which differs from the original operator by the additional term $\beta^2(\Delta t)^2\alpha_x\alpha_y L_{xx}L_{yy}\Delta T_{j,k}^{n+1}$. Thus the product approximates the operator to the left in equation (4.63) with accuracy $\approx O\left((\Delta t)^2\right)$.

If the product replaces the operator to the left in equation (4.63), the generalized two-layer (G2L) scheme given by

$$\left(1-\beta\Delta t\alpha_x L_{xx}\right)\left(1-\beta\Delta t\alpha_y L_{yy}\right)\Delta T_{j,k}^{n+1}=\Delta t\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^{n} \tag{4.64}$$

is applied at each time-step in the following two stages :

1. For an arbitrary line parallel to the axis Ox (k constant):

$$\left(1-\beta\Delta t\alpha_x L_{xx}\right)\Delta T_{j,k}^{*}=\Delta t\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^{n}\,. \tag{4.65}$$

2. For an arbitrary line parallel to the axis Oy (j constant):

$$\left(1-\beta\Delta t\alpha_y L_{yy}\right)\Delta T_{j,k}^{n+1}=\Delta T_{j,k}^{*}\,. \tag{4.66}$$

The highly economical Thomas algorithm executes both stages. The approximation accuracy for the G2L scheme is $\approx O\left((\Delta t)^2,(\Delta x)^2,(\Delta y)^2\right)$ for $\beta=0.5$. The scheme is absolutely stable for arbitrary $\beta\geq 0.5$. The discrepancy (4.47) required for the solution of stationary problems by the pseudo-nonstationary method is calculated for the G2L scheme in the course of finding the solution.

Remarks:

- *The advantage of G2L compared to the ADI scheme becomes apparent for the equations of a more complex structure than the MDDE, when most of the*

computation time is required for the right side of an equation that is calculated twice for the ADI, but only once for G2L;

- *G2L is generalizable to the three-dimensional case without any alteration.*

The generalized three-layer (G3L) scheme for the 2-D diffusion equation is

$$\frac{(1+\gamma)\Delta T_{j,k}^{n+1}}{\Delta t} - \frac{\gamma\Delta T_{j,k}^{n}}{\Delta t} = (1-\beta)\left(\alpha_x L_{xx} + \alpha_y L_{yy}\right)T_{j,k}^{n} + \\ +\beta\left(\alpha_x L_{xx} + \alpha_y L_{yy}\right)T_{j,k}^{n+1}, \tag{4.67}$$

where $\Delta T_{j,k}^{n} = T_{j,k}^{n} - T_{j,k}^{n-1}$. Applying the same approach to equation (4.67) as was used for the generation of G2L in (4.64)-(4.65) results in the following two-stage algorithm:

I. For an arbitrary line parallel to the coordinate axis *Ox* (*k* constant):

$$\left(1 - \frac{\beta}{(1+\gamma)}\Delta t\alpha_x L_{xx}\right)\Delta T_{j,k}^{*} = \frac{\Delta t}{(\gamma+1)}\left(\alpha_x L_{xx} + \alpha_y L_{yy}\right)T_{j,k}^{n} \tag{4.68}$$

II. For an arbitrary line parallel to the coordinate axis *Oy* (j constant):

$$\left(1 - \frac{\beta}{(1+\gamma)}\Delta t\alpha_y L_{yy}\right)\Delta T_{j,k}^{n+1} = \Delta T_{j,k}^{*}. \tag{4.69}$$

For the parameters $\beta = 1,\ \gamma = 0.5$, the two-stage algorithm, (4.68) and (4.69), approximates the two-dimensional diffusion equation (4.29) with accuracy $\approx O\left((\Delta t)^2, (\Delta x)^2, (\Delta y)^2\right)$; in addition, the algorithm is absolutely stable.

Remarks:

- *The numerical implementation of G3L uses G2L on the first step;*

- *The G3L scheme is generalized without change for the 3-D case.*

We consider a procedure for implementing the mixed boundary Dirichlet-Neumann conditions for the multi-dimensional diffusion equation by the use of splitting schemes. The first and third boundary conditions in (4.30) are left as they are, and the second and fourth boundary conditions are changed to

$$\frac{\partial T(1,y,t)}{\partial x} = g(y,t)\ ,\ \frac{\partial T(x,1,t)}{\partial y} = h(x,t)\,, \tag{4.70}$$

where $g(y,t), h(x,t)$ are known functions. The Neumann boundary conditions are made second-order accurate by using the following finite-difference approximation:

$$\frac{T_{j+1,\,k} - T_{j-1,\,k}}{2\Delta x} = g_k(t)\ ,\ \frac{T_{j,\,k+1} - T_{j,\,k-1}}{2\Delta y} = h_j(t)\ . \tag{4.71}$$

The FD approximation (4.71) is applied together with the generalized two-layer scheme.

For the points of the boundaries at $x = 1$ and $y = 1$, the right side of equation (4.65) is calculated by the first step of G2L by equation (4.71) after the calculation of two additional points $T_{j+1,k}$ and $T_{j,k+1}$, which lie outside of the numerical domain. But the right-hand operators of the equations (4.65) and (4.66) require the calculation of the inaccuracies $\Delta T^{*}_{j+1,\,k}$ and $\Delta T^{n+1}_{j,\,k+1}$, which can be done using equations (4.70) successively in time:

$$\Delta T^{n+1}_{j+1,\,k} = \Delta T^{n+1}_{j-1,\,k} + 2\Delta x \Delta g^{n+1}_{k}\,, \qquad \Delta T^{n+1}_{j,\,k+1} = \Delta T^{n+1}_{j,\,k-1} + 2\Delta y \Delta h^{n+1}_{j}\,, \tag{4.72}$$

where $\Delta g^{n+1}_{k} = g^{n+1}_{k} - g^{n}_{k}$, $\Delta h^{n+1}_{j} = h^{n+1}_{j} - h^{n}_{j}$.

Then, using equation (4.72) and modifying the corresponding components of the operators L_{xx}, L_{yy}, the two-dimensional operators have to be split into one-dimensional operators.

Thus, for $j = NX$, the first equation of (4.72) leads to the following equation, which changes on the first step of G2L, (4.65):

$$\begin{aligned}\left(1-\alpha_x \beta \Delta t L_{xx}^m\right)\Delta T_{j,k}^* &= \Delta t\left(\alpha_x L_{xx}+\alpha_y L_{yy}\right)T_{j,k}^n - \\ -2\Delta x\left(1-\alpha_x \beta \Delta t L_{xx}^m\right)&\left(1-\alpha_y \beta \Delta t L_{yy}\right)\Delta g_k^{n+1},\end{aligned} \tag{4.73}$$

where $L_{xx}^m = \{2,-2,0\}/(\Delta x)^2$. For the second step, equation (4.66) is used. For $k = NY$, equation (4.65) is unchanged on the first step. But the second step, instead of (4.66), yields

$$\left(1-\alpha_y \beta \Delta t L_{yy}^m\right)\Delta T_{j,k}^{n+1} = \Delta T_{j,k}^* - 2\Delta y\left(1-\alpha_y \beta \Delta t L_{yy}^m\right)\Delta h_j^{n+1}, \tag{4.74}$$

where $L_{yy}^m = \{0,-2,2\}^T/(\Delta y)^2$.

This procedure keeps the accuracy of the boundary condition approximation the same as that of the generalized two-layer (G2L) scheme at the internal points of the numerical domain.

4.3 Exercises on parabolic PDEs

1. Check the stability for a linear algebraic equation array with block tridiagonal matrix which can only be solved using implicit schemes.

2. The ends of an insulated rod of length 1m (see Fig. 4.1) at temperature $0^{o}C$ are placed in contact with hot reservoirs ($T=100^{o}C$) at $t=0$. Using the FTCS scheme, compute the rod temperature as a function of time for: $\alpha=10^{-5}\,\mathrm{m}^2/\mathrm{s},\ \Delta x=0.1,\ \Delta t=500\mathrm{s}$.

3. Solve problem 2 for $\Delta x=0.1,0.2$; $\Delta t=500\mathrm{s}$ and $s=0.5,0.3,0.1,1/6$. Analyze the dependence of the accuracy on the parameter *s*. *Hint:* consider T.E..

4. Redo problem 2 using the DuFort-Frankel scheme for $2\le t\le 9,\ \Delta x=0.05,0.1,0.2$ and $s=0.3,0.41,1/6,1/\sqrt{12}$. Compare with results obtained by the FTCS scheme.

5. Consider an infinite wall of thickness *L*=1m with initial temperature distribution $T(x)=c\sin(\pi x/L)$, for $0\le x\le 1$. If the wall temperature is kept at 0°C, the exact solution of the heat conduction equation (4.1) for $t>0$, $0\le x\le L$ is

$$T(t,x)=c\exp\left(\frac{-\alpha\pi^2 t}{L^2}\right)\sin\frac{\pi x}{L}.$$

Choose c = 100 °C, $\alpha=0.02\mathrm{m}^2/\mathrm{s}$ and compare the exact and numerical solutions of the problem using FTCS scheme for: a) $\Delta x=0.1,\ \Delta t=0.1$; b) $\Delta x=0.1,\ \Delta t=0.25$; c) $\Delta x=0.1,\ \Delta t=0.3$. Find when the solution loses stability and fails to convergence.

6. Repeat problem 5 on a finer grid in Δx. Prove whether or not an increase of the numerical solution accuracy corresponds to the order of approximation in Δx. Compare the results obtained with solution by the DuFort-Frankel numerical scheme.

7. Redo problem 5 using the absolutely stable alternating-direction explicit (ADE) method with accuracy $\approx O\left((\Delta t)^2,(\Delta x)^2\right)$ suggested by Barakat and Clark (1966) [2]. According to this method, the equations are solved by marching left-to-right, according to

$$\frac{p_j^{n+1}-p_j^n}{\Delta t}=\alpha\frac{p_{j-1}^{n+1}-p_j^{n+1}-p_j^n+p_{j+1}^n}{(\Delta x)^2}\ ,\ j=\overline{1,N-1}\ , \qquad \text{(E-4.1)}$$

and at the same time by marching right-to-left:

$$\frac{q_j^{n+1}-q_j^n}{\Delta t}=\alpha\frac{q_{j-1}^n-q_j^n-q_j^{n+1}+q_{j+1}^{n+1}}{(\Delta x)^2},\ j=\overline{N-1,1}\ . \qquad \text{(E-4.2)}$$

Moreover the solutions at the boundaries coincide with the known solutions for T_j^n. The solutions obtained are averaged to calculate

$$T_j^{n+1}=\frac{1}{2}\left(p_j^{n+1}+q_j^{n+1}\right)\ . \qquad \text{(E-4.3)}$$

8, 9. Redo problem 5 using: a) the Crank-Nicholson numerical scheme, b) the implicit generalized 3-layer (G3L) scheme for $\beta=0.5,\ \gamma=1.0$.

Optional tasks:

10. Solve problem 7 from the previous chapter by the pseudo-nonstationary method with a splitting procedure using the generalized implicit two-layer scheme.

11. Calculate (approximately) the number of operations required for the following numerical schemes: FI (fully implicit), Crank-Nicholson, G3L, FTCS and DuFort-Frankel. Compare their efficiency.

12. Show by von Neumann analysis that the FTCS scheme is conditionally stable for $s\le 0.5$.

5. Finite-difference algorithms for linear hyperbolic PDEs

In this and next chapters, we describe and analyse finite-difference schemes for the solution of several kinds of simple linear partial differential equation: the first-order wave equation, the stationary diffusion-convection equation, and the non-stationary convection-diffusion equation. All of these equations are called model equations, because they are used to study the behaviour of solutions to more complex partial differential equations. The model equations we will consider have analytical solutions for some initial and boundary conditions. Having these exact analytical solutions, one can easily estimate and compare different finite-difference schemes, which can then be used for the solution of more complex real problems. The methods described here have properties that are typical of classes of analogous methods. Some of these properties are undesirable, but they are nonetheless instructive.

5.1 One-dimensional wave equation

The one-dimensional wave equation,

$$\frac{\partial^2 \overline{T}}{\partial t^2} = c^2 \frac{\partial^2 \overline{T}}{\partial x^2} \tag{5.1}$$

is a hyperbolic PDE is which describes an acoustic wave spreading in a homogeneous medium with velocity c. Furthermore, the first-order equation,

$$\frac{\partial \overline{T}}{\partial t} + c\frac{\partial \overline{T}}{\partial x} = 0,\ c > 0, \tag{5.2}$$

has a solution similar to the solution of (5.1). Equation (5.1) is easily obtained from (5.2): first differentiate (5.2) by $\partial/\partial t$, then by $\partial/\partial x$, multiply the second equation by $-c$ and add

to the first one. Otherwise, equation (5.2) may be obtained by decomposing the differential operator in (5.1) through its factorization:

$$\frac{\partial^2}{\partial t^2} - c^2 \frac{\partial^2}{\partial x^2} = 0\,, \;\Rightarrow\; \left(\frac{\partial}{\partial t} - c\frac{\partial}{\partial x}\right)\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right) = 0\,.$$

Equation (5.2) is also a hyperbolic PDE and is called the one-dimensional diffusion equation. It has an exact analytical solution for the initial condition

$$\overline{T}(x,0) = F(x)\;,\; -\infty < x < +\infty\,, \tag{5.3}$$

and this solution is the following simple wave which spreads with velocity c along Ox:

$$\overline{T}(x,t) = F(x - ct). \tag{5.4}$$

Now let us study several finite-difference schemes for the solution of equation (5.2).

5.2 Methods of first and first/second-order accuracy

5.2.1 Explicit Euler method

This method consists of two simple explicit one-step schemes:

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} + c\frac{T_{j+1}^n - T_j^n}{\Delta x} = 0\,, \tag{5.5}$$

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} + c\frac{T_{j+1}^n - T_{j-1}^n}{2\Delta x} = 0\,, \tag{5.6}$$

with accuracy $\cong O(\Delta t, \Delta x)$ and $\cong O\left(\Delta t, (\Delta x)^2\right)$, respectively. Unfortunately, stability analysis shows both schemes to be absolutely unstable.

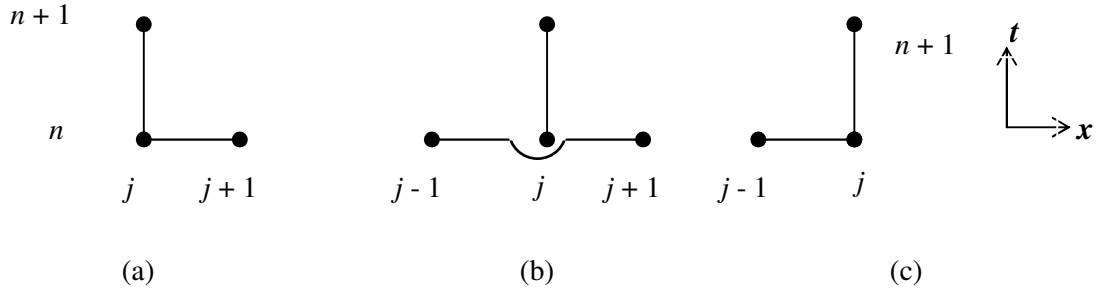


Fig. 5.1. Routines for the Euler method (a, b) and upwind scheme (c).

5.2.2 The simplest upwind scheme

Scheme (5.5) can be made stable if upwind, instead of forward, differencing is used for approximating the space derivative ($c > 0$):

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} + c\frac{T_j^n - T_{j-1}^n}{\Delta x} = 0\,. \tag{5.7}$$

For $c < 0$, stability is achieved by using forward differencing, similar to (5.5), instead. To get an estimate of the inaccuracy, a modified equation[#] is constructed for (5.7). For this, equation (5.7) uses, instead of T_j^{n+1} and T_{j-1}^n, the following Taylor series expansions:

$$\frac{1}{\Delta t}\left\{\left[T_j^n + \Delta t \quad T_t + \frac{(\Delta t)^2}{2}T_{tt} + \frac{(\Delta t)^3}{6}T_{ttt} + \ldots\right] - T_j^n\right\} +$$

$$+\frac{c}{\Delta x}\left\{T_j^n - \left[T_j^n - \Delta x \cdot T_x + \frac{(\Delta x)^2}{2}T_{xx} - \frac{(\Delta x)^3}{6}T_{xxx} + \ldots\right]\right\} = 0. \tag{5.8}$$

[#] In the Russian literature, the differential approximation of a finite-difference scheme is called the modified equation.

Now, some simple rearrangement of the terms in (5.8) yields

$$T_t + cT_x = -\frac{\Delta t}{2}T_{tt} + \frac{c\Delta x}{2}T_{xx} - \frac{(\Delta t)^2}{6}T_{ttt} - c\frac{(\Delta x)^2}{6}T_{xxx} + \ldots \tag{5.9}$$

Equation (5.9) contains the original wave equation on the left-hand side, and (evidently non-zero) approximation deficiency on the right. The significance of the terms on the right-hand side of equation (5.9) is better understood if the time derivatives are expressed in terms of the space derivatives. Differentiating equation (5.9) by t yields

$$T_{tt} + cT_{xt} = -\frac{\Delta t}{2}T_{ttt} + \frac{c\Delta x}{2}T_{xxt} - \frac{(\Delta t)^2}{6}T_{tttt} - \frac{c(\Delta x)^2}{6}T_{xxxt} + \ldots \tag{5.10}$$

Now, the second-order time derivative is expressed through the space derivatives. First equation (5.10) is differentiated by x and the result is then multiplied by $-c$. This leads to

$$-cT_{tx} - c^2T_{xx} = \frac{c\Delta t}{2}T_{ttx} - \frac{c^2\Delta x}{2}T_{xxx} + \frac{c(\Delta t)^2}{6}T_{tttx} + \frac{c^2(\Delta x)^2}{6}T_{xxxx} + \ldots \tag{5.11}$$

Adding equation (5.10) to (5.11) gives

$$T_{tt} = c^2T_{xx} + \Delta t\left(\frac{-T_{ttt}}{2} + \frac{c}{2}T_{ttx} + O(\Delta t)\right) + \Delta x\left(\frac{c}{2}T_{xxt} - \frac{c}{2}T_{xxx} + O(\Delta x)\right). \tag{5.12}$$

In a similar way, the following expressions can be derived:

$$\begin{aligned} T_{ttt} &= -c^3T_{xxx} + O(\Delta t, \Delta x); \\ T_{ttx} &= c^2T_{xxx} + O(\Delta t, \Delta x); \\ T_{xxt} &= -cT_{xxx} + O(\Delta t, \Delta x). \end{aligned} \tag{5.13}$$

Substituting equations (5.12) and (5.13) into (5.9) yields

$$T_t + cT_x = \frac{c\Delta x}{2}(1-\nu)T_{xx} - \frac{c(\Delta x)^2}{6}(2\nu^2 - 3\nu + 1)T_{xxx} + \\ +O\left((\Delta x)^3;(\Delta x)^2\Delta t;\Delta x(\Delta t)^2;(\Delta t)^3\right) \ , \tag{5.14}$$

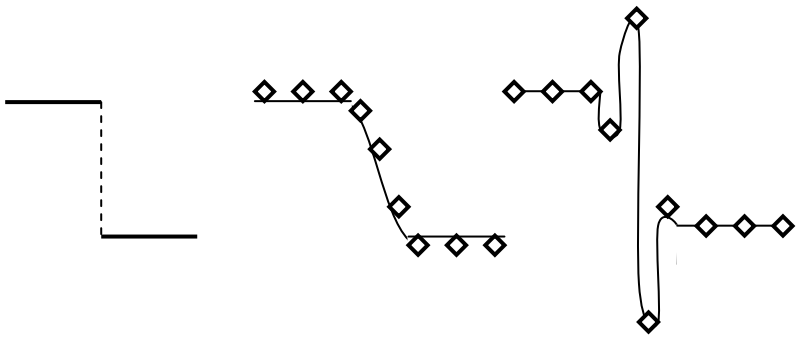
where $\nu = c\Delta t/\Delta x$. An equation of type (5.14) is called the modified equation. Note that this equation is really solved by replacing equation (5.2) by its finite-difference approximation. The right-hand side of equation (5.14) is the approximation deficiency, which indicates that the lowest-order term on the right-hand side of the modified equation (5.14) predetermines the method's order of accuracy. This term in (5.14) is also called the truncation error (T.E.).

If $\nu = 1$, then the right-hand side of (5.14) is identically equal to zero (T.E. $\equiv 0$) and the solution of the finite-difference equation is the exact solution of the original differential equation (5.2). In this case, the following scheme is obtained:

$$T_j^{n+1} = T_{j-1}^n. \tag{5.15}$$

Solution by this scheme coincides with the exact solution of the original equation by the method of characteristics, which will be described later.

For $\nu \neq 1$, the main term in the approximation deficiency is proportional to the second-order space derivative T_{xx} , so that it is similar to a dissipative viscous term in one-dimensional fluid flow. This behaviour of the finite-difference scheme is conditioned by the even-order derivatives in the deficiency formula and is called the dissipation on a finite-difference grid. The scheme viscosity smoothes the solution, in the sense of decreasing the gradients of all the variables, e.g. instead of the stepped exact solution (a) showed in Fig. 5.2, the smooth function (b) will be obtained. When the deficiency is mainly dissipative, the numerical solution is typical of one for numerical schemes having first-order accuracy.



(a) (b) (c)

Fig. 5.2. Influence of dissipation (b) and dispersion (c) on the exact solution (a) of the diffusion equation.

Another (close to the physical) property of the finite-difference schemes is dispersion, which is caused by odd derivatives in the truncation error (T.E.). Dispersion destroys the correlation between wave phases, as is observed in Fig. 5.2 (c).

Remarks:

- *If a finite-difference scheme permits an exact solution of the equation, the scheme is said to satisfy the "shift condition".*
- *For $\nu \neq 1$, a scheme with upwind differencing introduces additional artificial (numerical) viscosity into the original equation; this is called implicit artificial viscosity, in contrast to explicit artificial viscosity, which is sometimes introduced intentionally.*

5.2.3 Lax scheme

The finite-difference scheme in (5.6) can be also transformed by the Euler method to a stable one if T_j^n is replaced by the space-averaged value $\left(T_{j+1}^n + T_{j-1}^n\right)/2$. This gives the Lax scheme,

$$\frac{T_j^{n+1}-\left(T_{j+1}^n+T_{j-1}^n\right)/2}{\Delta t}+c\frac{T_{j+1}^n-T_{j-1}^n}{2\Delta x}=0, \tag{5.16}$$

which has accuracy $\cong O\left(\Delta t,(\Delta x)^2/\Delta t\right)$ and is stable for $|\nu|\le 1$. The modified equation for the Lax scheme is

$$T_t+cT_x=\frac{c\Delta x}{2}\left(\frac{1}{\nu}-\nu\right)T_{xx}+\frac{c(\Delta x)^2}{3}\left(1-\nu^2\right)T_{xxx}+\dots, \tag{5.17}$$

from which it follows that the Lax scheme satisfies the "shift condition" for $\nu=1$, and that for $\nu\neq 1$ it has a high level of dissipation.

5.2.4 Implicit Euler method

Now, consider the following implicit scheme:

$$\frac{T_j^{n+1}-T_j^n}{\Delta t}+\frac{c}{2\Delta x}\left(T_{j+1}^{n+1}-T_{j-1}^{n+1}\right)=0. \tag{5.18}$$

This scheme has approximation accuracy $\cong O\left(\Delta t,(\Delta x)^2\right)$ and is absolutely stable. As was stated for implicit schemes earlier, the transition to the next time-step is performed after the solution of a linear algebraic equation array with block tridiagonal structure by the Thomas marching algorithm:

$$\frac{\nu}{2}T_{j+1}^{n+1}+T_j^{n+1}-\frac{\nu}{2}T_{j-1}^{n+1}=T_j^n. \tag{5.19}$$

The truncation error (T.E.) for the implicit Euler scheme is

$$\text{T.E.}=\left(\frac{1}{2}c^2\Delta t\right)T_{xx}-\left[\frac{1}{6}c(\Delta x)^2+\frac{1}{3}c^3(\Delta t)^2\right]T_{xxx}+\dots,$$

which shows the dissipative character of the solution obtained.

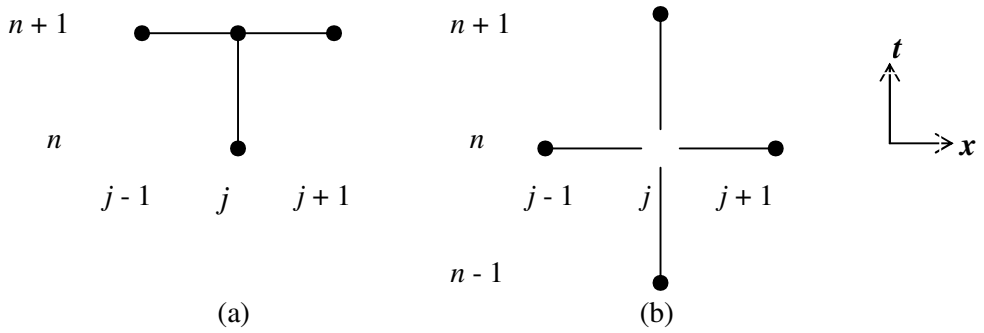


Fig. 5.3. Routines for the implicit Euler method (a) and the "leap-frog" method (b).

5.3 Methods of second-order accuracy

Now we consider completely second-order accurate numerical schemes. These schemes are popular due to their comparatively high accuracy and reasonable computational cost.

5.3.1 Skip method ("leap-frog")

The skip method is the simplest explicit three-layer method having second-order accuracy. The finite-difference approximation for the method is:

$$\frac{T_j^{n+1} - T_j^{n-1}}{2\Delta t} + c\frac{T_{j+1}^n - T_{j-1}^n}{2\Delta x} = 0\,. \tag{5.20}$$

The accuracy of the method is $\cong O\left((\Delta t)^2, (\Delta x)^2\right)$. It is stable for $|\nu| \le 1$, and its truncation error is given by

$$\text{T.E.} = \frac{c(\Delta x)^2}{6}\left(\nu^2 - 1\right)T_{xxx} - \frac{c(\Delta x)^4}{120}\left(9\nu^4 - 10\nu^2 + 1\right)T_{xxxx} + \ldots\,,$$

which indicates that the scheme has mainly dispersive properties.

Remarks:

- *In the first stage of the "leap-frog" method, the two-step method should be used (first, initial conditions must be specified at two-time levels).*
- *The "skip" leads to two independent solutions, and the final solution is therefore calculated by averaging over the neighboring nodes.*
- *The additional computer storage required for three-layer schemes may be reduced considerably by a simple overwriting procedure, whereby* T_j^{n-1} *is overwritten by* T_j^{n+1}.
- *For* $\nu = \pm 1$, *the "leap-frog" method satisfies the "shift condition".*

5.3.2 The Lax-Wendroff scheme

The Lax-Wendroff finite-difference scheme is based on the Taylor series expansion

$$T_j^{n+1} = T_j^n + \Delta t \cdot T_t + \frac{1}{2}(\Delta t)^2 T_{tt} + O\left((\Delta t)^3\right). \tag{5.21}$$

From the wave equation,

$$T_t = -cT_x,\ T_{tt} = c^2 T_{xx}; \tag{5.22}$$

after substitution of the equations in (5.22) into the finite-difference equation (5.21), we have

$$T_j^{n+1} = T_j^n - c\Delta t T_x + \frac{1}{2}c^2(\Delta t)^2 T_{xx} + O\left((\Delta t)^3\right). \tag{5.23}$$

Now, replacing T_x and T_{xx} by their central differences gives

$$T_j^{n+1} = T_j^n - \frac{c\Delta t}{2\Delta x}\left(T_{j+1}^n - T_{j-1}^n\right) + \frac{c^2(\Delta t)^2}{2(\Delta x)^2}\left(T_{j+1}^n - 2T_j^n + T_{j-1}^n\right). \tag{5.24}$$

The scheme accuracy is $\cong O\left((\Delta t)^2, (\Delta x)^2\right)$, and the scheme is stable for $|\nu| \le 1$.

The leading term of the truncation error (T.E.) for the Lax-Wendroff scheme is

$$\text{T.E.} = -c\frac{(\Delta x)^2}{6}\left(1-\nu^2\right)T_{xxx} - \frac{c(\Delta x)^3}{8}\nu\left(1-\nu^2\right)T_{xxxx} + \dots ,$$

which indicates the dispersive character of the approximation inaccuracy.

5.3.3 Two-step Lax-Wendroff scheme

This method is a modification of the one considered above and is applied most often for the solution of non-linear equations:

Step 1 is the Lax method for the solution of the problem on the first half-step in time:

$$\frac{T_{j+1/2}^{n+1/2} - \left(T_{j+1}^{n} + T_{j}^{n}\right)/2}{\Delta t/2} + c\frac{T_{j+1}^{n} - T_{j}^{n}}{\Delta x} = 0, \tag{5.25a}$$

Step 2 is the skip method applied on the second half-step in time:

$$\frac{T_{j}^{n+1} - T_{j}^{n}}{\Delta t} + c\frac{T_{j+1/2}^{n+1/2} - T_{j-1/2}^{n+1/2}}{\Delta x} = 0. \tag{5.25b}$$

This scheme also has approximation accuracy $\cong O\left((\Delta x)^2, (\Delta t)^2\right)$ and is stable for $|\nu| \le 1$.

5.3.4 McCormack predictor-corrector scheme

This numerical scheme is a version of the two-step Lax-Wendroff finite-difference scheme considered above, but without a computation of the function at the points $j+1/2$ and $j-1/2$. The algorithm for the method also consists of two steps:

Predictor:

$$\overline{T_j^{n+1}} = T_j^n - c\frac{\Delta t}{\Delta x}\left(T_{j+1}^n - T_j^n\right), \tag{5.26a}$$

Corrector:

$$T_j^{n+1} = \frac{1}{2}\left[T_j^n + \overline{T_j^{n+1}} - c\frac{\Delta t}{\Delta x}\left(\overline{T_j^{n+1}} - \overline{T_{j-1}^{n+1}}\right)\right]. \tag{5.26b}$$

The stability condition and the method accuracy coincide with those for the two-step Lax-Wendroff method.

Remark: *The derivative $\partial T/\partial x$ in the predictor step is approximated by a forward finite-difference, and in the corrector step by an upwind finite-difference.*

5.3.5 The upwind differencing scheme

The upwind differencing scheme proposed by Beam and Warming [7] becomes the modified McCormack predictor-corrector finite-difference scheme when upwind differences are used for the derivative $\partial T/\partial x$ on the predictor step as well as on the corrector step. The algorithm is the following:

Predictor:

$$\overline{T_j^{n+1}} = T_j^n - \frac{c\Delta t}{\Delta x}\left(T_j^n - T_{j-1}^n\right). \tag{5.27a}$$

Corrector:

$$T_j^{n+1} = \frac{1}{2}\left[T_j^n + \overline{T_j^{n+1}} - \frac{c\Delta t}{\Delta x}\left(\overline{T_j^{n+1}} - \overline{T_{j-1}^{n+1}}\right) - \frac{c\Delta t}{\Delta x}\left(T_j^n - 2T_{j-1}^n + T_{j-2}^n\right)\right]. \tag{5.27b}$$

The scheme has accuracy $\cong O\left((\Delta t)^2, (\Delta t\ \ \Delta x), (\Delta x)^2\right)$, but for $\nu = 1$ and $\nu = 2$ it satisfies the "shift condition", i.e. the scheme has infinite-order accuracy.

The truncation error (T.E.) for this scheme has the form

$$\text{T.E.} = \frac{c(\Delta x)^2}{6}(1-\nu)(2-\nu)T_{xxx} - \frac{(\Delta x)^4}{8\Delta t}\nu(1-\nu)^2(2-\nu)T_{xxxx} + \ldots$$

The von Neumann condition shows that this finite-difference scheme is stable for $|\nu| \le 2$.

***Remark**: For $|\nu| < 1$, the two-step Lax-Wendroff method and the upwind differencing method have conflicting phase discrepancies; consequently, the dispersion can be substantially reduced applying a linear combination of these methods, one of which is the well-known Fromm method [22, 23].*

5.3.6 The central time differencing implicit scheme (CTI)

For the construction of a second-order implicit finite-difference scheme, we consider here the subtraction of the two Taylor series expansions,

$$T_j^{n+1} = T_j^n + \Delta t\,(T_t)_j^n + \frac{(\Delta t)^2}{2}(T_{tt})_j^n + \frac{(\Delta t)^3}{6}(T_{ttt})_j^n + \ldots$$

-

$$T_j^n = T_j^{n+1} - \Delta t\,(T_t)_j^{n+1} + \frac{(\Delta t)^2}{2}(T_{tt})_j^{n+1} - \frac{(\Delta t)^3}{6}(T_{ttt})_j^{n+1} + \ldots$$

followed by the substitution: $(T_{tt})_j^{n+1} = (T_{tt})_j^n + \Delta t\,(T_{ttt})_j^n + \ldots$; this yields

$$T_j^{n+1} = T_j^n + \frac{\Delta t}{2}\left[(T_t)^n + (T_t)^{n+1}\right]_j = O\left((\Delta t)^3\right). \tag{5.28}$$

Expression (5.28) for a time derivative is called the finite-difference Crank-Nicholson approximation. Substituting $T_t = -cT_x$, we obtain

$$T_j^{n+1} = T_j^n - \frac{c\Delta t}{2}\left[(T_x)^n + (T_x)^{n+1}\right]_j + O\left((\Delta t)^3\right) \tag{5.29}$$

Now, with a central difference approximation for T_x, it follows that

$$T_j^{n+1} = T_j^n - \frac{\nu}{4}\left(T_{j+1}^{n+1} + T_{j+1}^n - T_{j-1}^{n+1} - T_{j-1}^n\right). \tag{5.30}$$

The scheme has approximation accuracy $\cong O\left((\Delta x)^2, (\Delta t)^2\right)$ and is absolutely stable. The truncation error is

$$\text{T.E.} = -\left[\frac{c^3(\Delta t)^2}{12} + \frac{c(\Delta x)^2}{6}\right]T_{xxx} - \left[\frac{c(\Delta x)^4}{120} + \frac{c^3(\Delta t)^2(\Delta x)^2}{24} + \frac{c^4(\Delta t)^4}{80}\right]T_{xxxxx} + \ \ldots, \tag{5.31}$$

which indicates the general dispersive character of the inaccuracy.

Remark: *In the formula for T.E. (5.31), the even-order derivatives are absent, meaning that the implicit artificial viscosity is identically zero. Thus, when using this scheme, a "smoothing" term in the form of an <u>explicit</u> "artificial" viscosity is often introduced to avoid nonlinear instability.*

For the central time differencing implicit scheme (CTI), it is possible to achieve even a fourth-order space approximation if the following correlation is used for T_x:

$$(T_x)_j = \frac{1}{2\Delta x}\frac{\overline{\delta}_x}{1 + \hat{\delta}_x^2/6}T_j + O\left((\Delta x)^4\right), \tag{5.32}$$

where

$$\overline{\delta}_x T_j \equiv L_x T_j = T_{j+1} - T_{j-1}, \qquad \hat{\delta}_x^2 \equiv L_{xx} = \overline{\delta}_x\left(\overline{\delta}_x T_j\right) = T_{j+1} - 2T_j + T_{j-1}.$$

Now let $T_x \equiv V$. Then, from (5.32),

$$\left(1 + \hat{\delta}_x^2/6\right)V_j = \overline{\delta}_x T_j/(2\Delta x),$$

or, rewritten in another form:

$$\frac{1}{6}\left(V_{j+1}+4V_j+V_{j-1}\right)=\frac{\bar{\delta}_x T_j}{2\Delta x}. \tag{5.33}$$

The finite-difference equation (5.33) thus obtained is implicit with respect to the derivative T_x. To calculate the derivative T_x, one therefore needs to solve a linear algebraic equation array with a block tridiagonal structure by the Thomas marching method using the known values of T_j.

5.4 The third-order accurate Rusanov method

Up until now, we have considered only first- or second-order accurate methods. Less common in the literature are third-order accurate methods, although one such is an explicit three-step method proposed by Rusanov [20]:

$$\underline{\text{Step 1}}: \qquad T^{(1)}_{j+½}=\frac{1}{2}\left(T^n_{j+1}+T^n_j\right)-\frac{1}{3}\nu\left(T^n_{j+1}-T^n_j\right)\ ;$$

$$\underline{\text{Step\_2}}: \qquad T^{(2)}_j=T^n_j-\frac{2}{3}\nu\left(T^{(1)}_{j+½}-T^{(1)}_{j-½}\right)\ ; \tag{5.34}$$

$$\underline{\text{Step\_3}}: \quad T^{n+1}_j=T^n_j-\frac{1}{24}\nu\left(-2T^n_{j+2}+7T^n_{j+1}-7T^n_{j-1}+2T^n_{j-2}\right)-\frac{3}{8}\nu\left(T^{(2)}_{j+1}-T^{(2)}_{j-1}\right)-\frac{\omega}{24}\hat{\delta}^4_x T^n_j,$$

where $\hat{\delta}^4_x T^n_j$ is the fourth-order finite-difference operator

$$\hat{\delta}^4_x T^n_j=T^n_{j+2}-4T^n_{j+1}+6T^n_j-4T^n_{j-1}+T^n_{j-2}\ .$$

Here, the term $\hat{\delta}^4_x T^n_j$ is needed for the stability condition

$$|\nu| \le 1,\ 4\nu^2 - \nu^4 \le \omega \le 3. \tag{5.35}$$

Obviously if $\omega \equiv 0$, the term with $\widehat{\delta}_x^4 T_j^n$ is absent and the scheme appears to be unstable.

The truncation error for this scheme is:

$$\text{T.E.} = -\frac{c(\Delta x)^3}{24}\left(\frac{\omega}{\nu} - 4\nu + \nu^3\right)T_{xxxx} + \frac{c(\Delta x)^4}{120}\left(-5\omega + 4 + 15\nu^2 - 4\nu^4\right)T_{xxxxx} + \ldots \tag{5.36}$$

Remarks:

- *Formula (5.36) shows that the dissipative properties of the difference scheme can be reduced if*

  $$\omega = 4\nu^2 - \nu^4, \tag{5.37}$$

  and that the dispersive properties can be reduced if the term multiplying T_{xxxxx} *is zero, i.e. if*

  $$\omega = \left(4\nu^2 + 1\right)\left(4 - \nu^2\right)/5. \tag{5.38}$$

- *When solving hyperbolic equations, explicit methods are preferred to implicit ones, because solutions are non-stationary and it is often of interest to find them for small time intervals,* Δt.

6. FDAs for convection-diffusion and transport equations

Convection-diffusion and more general transport equations describe a variety of different physical processes, including many in continuum mechanics. Hence, it is important to understand the basic properties of these equations and the numerical methods used to solve them. Generally speaking, these processes are described by parabolic or hyperbolic PDEs. This chapter is therefore devoted to finite-difference approximations (FDA) for convection-diffusion and transport equations.

6.1 Stationary FD convection-diffusion problems

First, we consider stationary boundary-value problems for convection-diffusion processes, starting with analysis for the simplest finite-difference (FD) schemes for the numerical solution of convection and diffusion problems.

6.1.1 Simplest FD approximations. Grid Reynolds number

For circumfluent problems, dissipation mechanisms are substantial only in a narrow layer, normally near the streamlined body (boundary layer). Numerical solutions on finite-difference grids adapted to the main flow often oscillate in the boundary-layer region. Here, an appropriate model equation to describe the stationary balance between convection and diffusion processes is:

$$u\frac{d\bar{T}}{dx}-\alpha\frac{d^2\bar{T}}{dx^2}=0\ . \tag{6.1}$$

For the boundary conditions

$$\bar{T}(0)=0, \qquad \bar{T}(1)=1, \tag{6.2}$$

the exact analytical solution of the boundary-value problem (6.1) and (6.2) in the interval $0 \le x \le 1$ is (see Fig. 6.1) :

$$\bar{T}(x)=\frac{e^{ux/\alpha}-1}{e^{u/\alpha}-1} \tag{6.3}$$

The solution is nearly constant in the whole interval [0,1), although it increases abruptly in the neighbourhood of $x=1$.

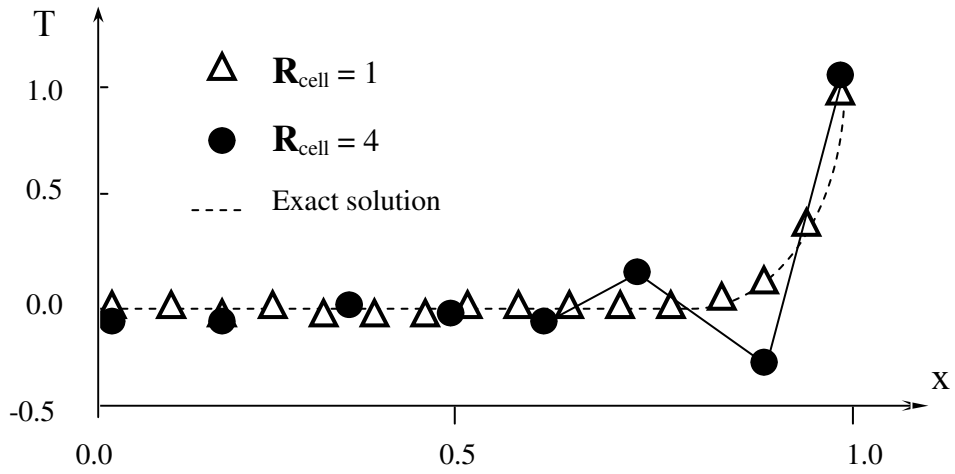


Fig. 6.1. Dependence of the solution of a convection-diffusion problem on grid Reynolds number.

Using central differencing for the approximation of equation (6.1) yields

$$-(1+0.5\mathbf{R}_{cell})T_{j-1}+2T_j-(1-0.5\mathbf{R}_{cell})T_{j+1}=0, \tag{6.4}$$

where $\mathbf{R}_{cell} = u\Delta x/\alpha$ is the grid Reynolds number. Here, the approximation accuracy is $\cong O\left((\Delta x)^2\right)$. The solution of the linear algebraic equation array for the boundary conditions in (6.2) is shown in Fig. 6.1 for different grid Reynolds numbers, $\mathbf{R}_{cell}$. From the stability condition for the solution of the equation array (6.4), which requires the matrix diagonal dominance, it becomes obvious that there are no oscillations in the solution if $\mathbf{R}_{cell} \le 2$. Moreover, this condition is also subject to the constraint that the eigenvalues of the matrix of a linear algebraic equation array (LAEA) should be <u>real</u>.

Using upwind differencing in the form $\left(T_j - T_{j-1}\right)/\Delta x$ for the approximation of the $\partial\bar{T}/\partial x$ term in equation (6.1) results in a solution that is free of oscillations. In this case, the solution algorithm is

$$-\left(1+\mathbf{R}_{cell}\right)T_{j-1} + 2\ \left(1+0.5\mathbf{R}_{cell}\right)T_j - T_{j+1} = 0. \tag{6.5}$$

However, the approximation accuracy for equation (6.1) becomes now $\cong O(\Delta x)$. Note that in this case the matrix of the LAEA has only real eigenvalues, ensuring that the solution will always converge.

Remarks:

- *The Taylor series expansion for equation (6.5) shows that it approximates the equation*

$$u\frac{d\bar{T}}{dx} - \alpha\left(1+0.5\mathbf{R}_{cell}\right)\frac{d^2\bar{T}}{dx^2} = 0 \tag{6.6}$$

  with accuracy $\cong O(\Delta x)^2$. That is to say, upwind differencing leads to the appearance of artificial diffusion $0.5\mathbf{R}_{cell}$, similar to the upwind differencing in the convection equation. The condition for the existence of an exact solution for equation (6.6), namely $\underline{\underline{0.5\mathbf{R}_{cell} << 1}}$, is stronger than that for equation (6.1).

- *If a Neumann (rather than Dirichlet) boundary condition is prescribed at* $x = 1$*, the solution character does not change substantially, although for a larger* u/α *ratio the central differencing scheme leads to even greater oscillations.*

6.1.2 Higher-order upwind differencing schemes

The character of the solutions obtained for the approximations of equation (6.1) considered above leads to the notion that a four-point approximation of the space derivative $\partial\overline{T}/\partial x$ should be used for the exact analytical solution. Thus, for $u > 0$:

$$L_x^{(4)}T \equiv \frac{T_{j+1} - T_{j-1}}{2\Delta x} + \frac{q\left(T_{j-2} - 3T_{j-1} + 3T_j - T_{j+1}\right)}{3\Delta x} + O\left((\Delta x)^2\right). \tag{6.7}$$

For $u < 0$, this becomes:

$$L_x^{(4)}T \equiv \frac{T_{j+1} - T_{j-1}}{2\Delta x} + \frac{q\left(T_{j-1} - 3T_j + 3T_{j+1} - T_{j+2}\right)}{3\Delta x} + O\left((\Delta x)^2\right). \tag{6.8}$$

A Taylor series expansion in the neighbourhood of the *j*-th node results in:

$$T_{j-2} - 3T_{j-1} + 3T_j - T_{j+1} \equiv \left[-(\Delta x)^3 T_{xxx} + 0.5(\Delta x)^4 T_{xxxx} + \ldots\right]_j; \tag{6.9}$$

that is, for $q = 0.5$, the term containing $(\Delta x)^2 T_{xxx}$ is eliminated from scheme (6.7), and the numerical scheme becomes third-order accurate. Applying the finite-difference scheme (6.7) constructed above for the approximation of equation (6.1) leads to the following numerical algorithm,

$$\frac{q}{3}\mathbf{R}_{cell}T_{j-2} - \left[1 + (q + 0.5)\mathbf{R}_{cell}\right]T_{j-1} + (2 + q\mathbf{R}_{cell})T_j - \left[1 + \left(\frac{q}{3} - 0.5\right)\mathbf{R}_{cell}\right]T_{j+1} = 0, \tag{6.10}$$

which is an LAEA with a tetradiagonal matrix that can be solved by the generalized Thomas algorithm.

The following differential equation is equivalent to the finite-difference equation (6.10):

$$T_x - \frac{\Delta x}{\mathbf{R}_{cell}} T_{xx} + \underline{(1-2q)\frac{(\Delta x)^2}{6} T_{xxx}} + \left(2q - \frac{1}{\mathbf{R}_{cell}}\right)\frac{(\Delta x)^3}{12} T_{xxxx} + \\ + (1-10q)\frac{(\Delta x)^4}{120} T_{xxxxx} + \ldots = 0. \tag{6.11}$$

Remarks:

- *In practice, a small ratio α/u is typical, and therefore usually $\mathbf{R}_{cell} \cong O(1)$ or $\mathbf{R}_{cell} > O(1)$. The latter means that the approximation accuracy in (6.1) may influence the approximation accuracy (6.10) in different ways. For example, for $q \neq 0.5$ the term with T_{xxx} is the most important one in equation (6.11). Thus, for problems with a small diffusion coefficient, the convective term $\underline{uT_x}$ should be approximated more precisely than the diffusion term $\underline{\alpha T_{xx}}$.*

- *The analysis shows that the odd-order space derivatives in the approximation accuracy are mainly responsible for the solution oscillations, while the even-order ones are mainly associated with the dissipation or unlimited increase of the solution, depending on the sign of the factor in front of the derivative.*

6.2 One-dimensional transport equation

The one-dimensional transport equation is written in the form

$$\frac{\partial \overline{T}}{\partial t} + u\frac{\partial \overline{T}}{\partial x} - \alpha\frac{\partial^2 \overline{T}}{\partial x^2} = 0, \tag{6.12}$$

where $\overline{T}$ is a passive scalar property available for convection with velocity $u(x,t)$ and for diffusion with an intensity expressed by the diffusion coefficient α. To begin with, assume u and α are constants.

Obviously, equation (6.12) is strictly parabolic and requires, in the same way as the diffusion equation, either Dirichlet boundary conditions in the form

$$\bar{T}(0,t) = a(t) \ , \ \bar{T}(1,t) = b(t), \tag{6.13a}$$

or Neumann boundary conditions in the form

$$\frac{\partial \bar{T}}{\partial x}(0,t) = c(t) \ , \ \frac{\partial \bar{T}}{\partial x}(1,t) = d(t), \tag{6.13b}$$

as well as an initial condition,

$$\bar{T}(x,0) = T_0(x). \tag{6.13c}$$

For large u/α ratios, however, we can expect that the first two terms in equation (6.12) will predominate, in which case the equation, now convection-dominated, becomes hyperbolic. Because exact solutions of the convection equation are normally non-decreasing spreading waves (i.e. waves without amplitude decrease), for large u/α ratios the solution of equation (6.12) is expected to be a slightly decreasing wave.

Based on the previous analysis of the stationary diffusion-convection equation, which is the limiting case of equation (6.12), it is also to be expected that, for approximate solutions of equation (6.12), spatial waves may occur for large u/α ratios if symmetric three-point algebraic approximations are used for the convective term.

Now the algebraic approximation schemes (FTCS, DuFort-Frankel, etc.), which were considered earlier for the diffusion and convection equations separately, will be analyzed and studied to investigate the numerical complications that arise when diffusion and convection occur simultaneously.

6.2.1 Explicit FD schemes

Now consider some explicit approaches to the Cauchy problem. Using the FTCS (forward in time, central in space) scheme for the approximation of equation (6.12) leads to

$$\frac{T_j^{n+1}-T_j^n}{\Delta t}+\frac{u\left(T_{j+1}^n-T_{j-1}^n\right)}{2\Delta x}-\frac{\alpha\left(T_{j-1}^n-2T_j^n+T_{j+1}^n\right)}{(\Delta x)^2}=0, \tag{6.14}$$

which, in turn, results in the numerical algorithm

$$T_j^{n+1}=\left(S+0.5C\right)T_{j-1}^n+\left(1-2S\right)T_j^n+\left(S-0.5C\right)T_{j+1}^n, \tag{6.15}$$

where $S=\alpha\Delta t/\Delta x^2$, $C=u\Delta t/\Delta x$. From a Taylor series expansion in the neighborhood of the node (j,n), it follows that the numerical scheme given by (6.15) approximates equation (6.12) with accuracy $\cong O\left(\Delta t,(\Delta x)^2\right)$, and equation

$$\frac{\partial\overline{T}}{\partial t}+u\frac{\partial\overline{T}}{\partial x}-\alpha\frac{\partial^2\overline{T}}{\partial x^2}+\underline{\underline{\frac{\Delta t}{2}\frac{\partial^2\overline{T}}{\partial t^2}}}=0, \tag{6.16}$$

with accuracy $\cong O\left((\Delta t)^2,(\Delta x)^2\right)$. In equation (6.16), the underlined term can be interpreted as the leading term of the approximation inaccuracy. Replacing the second-order time derivative by spatial derivatives as before, we get

$$\begin{aligned}&\frac{\partial\overline{T}}{\partial t}+u\frac{\partial\overline{T}}{\partial x}-\left(\alpha-\alpha'\right)\frac{\partial^2\overline{T}}{\partial x^2}-\left(\alpha u\Delta t+u^3\frac{(\Delta t)^2}{3}-u\frac{(\Delta x)^2}{6}\right)\frac{\partial^3\overline{T}}{\partial x^3}+\\&+\left(\alpha^2\frac{\Delta t}{2}-\alpha u^2(\Delta t)^2+u^4\frac{(\Delta t)^3}{4}-\alpha\frac{(\Delta x)^2}{12}+u^2\frac{\Delta t}{6}(\Delta x)^2\right)\frac{\partial^4\overline{T}}{\partial x^4}=0,\end{aligned} \tag{6.17}$$

where $\alpha' = u^2 \Delta t/2$. For large u/α ratios, the term with artificial diffusion[#] $\alpha' \partial^2 \overline{T}/\partial x^2$ can be of the same order of magnitude as the natural diffusion $\alpha \partial^2 \overline{T}/\partial x^2$ if no restriction on the time step Δt is imposed. Thus, the application of a scheme that is first-order accurate in time causes first-order dissipation and dispersion in the transport equation. If the term $\partial^4 \overline{T}/\partial x^4$ is not too large, then the last term in equation (6.17) is not substantial. Therefore the restriction of the time step Δt results in the following stability condition for the numerical scheme:

$$\Delta t << 2\alpha/u^2, \quad \text{or} \quad C^2 << 2S, \quad \text{or} \quad \mathbf{R}_{cell} \left(\equiv C/S \right) << 2/C. \tag{6.18}$$

Remarks:

- *The conditions in (6.18) follow give good reason to use numerical schemes having* $\cong O\left((\Delta t)^2, ... \right)$ *accuracy for the transport equation.*

- *The FTCS scheme is conditionally stable for the diffusion equation and is <u>always unstable</u> for the convection equation; therefore, for the transport equation, conditional stability is to be expected for*

  $$0 \le C^2 \le 2S \le 1, \tag{6.19}$$

  while (6.19) admits the existence of a solution for $\mathbf{R}_{cell} = u\Delta x/\alpha = C/S > 2$, *i.e. in this case oscillating solutions are expected.*

The DuFort-Frankel scheme for equation (6.12) results in the approximation

$$\frac{T_j^{n+1} - T_j^{n-1}}{2\Delta t} + u \frac{T_{j+1}^n - T_{j-1}^n}{2\Delta x} - \frac{\alpha \left[T_{j-1}^n - \left(T_j^{n-1} + T_j^{n+1} \right) + T_{j+1}^n \right]}{(\Delta x)^2}. \tag{6.20}$$

Stability analysis shows that, for $C = u\Delta t/\Delta x \le 1$, additional restrictions on S are not required.

[#] for more details about artificial diffusion, see the papers of Henshaw, etc. [27, 28]

The truncation error (T.E.) for the approximation inaccuracy in the DuFort-Frankel scheme has the following form

$$\text{T.E.} = \alpha C^2 T_{xx} + \left(1 - C^2\right)\left[\frac{u(\Delta x)^2}{6} - \frac{2\alpha^2 \quad C^2}{u}\right] T_{xxx} + \ldots, \tag{6.21}$$

which requires the condition $\Delta t << \Delta x$ to be satisfied and, more precisely, $C^2 << 1$ - a very strict restriction.

If upwind differencing is used instead of central differencing in the FTCS scheme for the approximation of $\partial \overline{T}/\partial x$, then for $u > 0$ the following algorithm is obtained:

$$T_j^{n+1} = (S + C)T_{j-1}^n + (1 - 2S - C)T_j^n + ST_{j+1}^n. \tag{6.22}$$

The algebraic equation in (6.22) approximates the differential equation in (6.12) with accuracy $\cong O(\Delta t, \Delta x)$ while the expression for the approximation inaccuracy gives

$$\text{T.E.} = -\frac{u\Delta x}{2}(1 - C)T_{xx} - \left[C\alpha\Delta x - \frac{u(\Delta x)^2}{6}(1 - 3C + 2C^2)\right] T_{xxx} + \ldots \tag{6.23}$$

Thus, the scheme creates artificial diffusion with the associated coefficient being $\alpha' = 0.5u\Delta x(1 - C)$. Note that the same term occurs also in the case of the transport equation, and the required approximation accuracy is guaranteed here for the following conditions:

$$\alpha' << \alpha, \quad \text{or} \quad \mathbf{R}_{cell} << 2/(1 - C). \tag{6.24}$$

A direct analysis of the scheme stability by von Neumann results in the condition

$$C + 2S \le 1. \tag{6.25}$$

From (6.25) follows a restriction on the time step,

$$\Delta t \le 0.5(\Delta x)^2 \big/ (1 + 0.5\mathbf{R}_{cell}), \tag{6.26}$$

which is much stronger than that for the diffusion equation.

The Lax-Wendroff scheme keeps the approximation for the original equation (6.12) second-order accurate, both in time as well as space:

$$\frac{\Delta T_j^{n+1}}{\Delta t} + uL_x T_j^n - \alpha^* L_{xx} T_j^n = 0, \tag{6.27}$$

where

$$\Delta T_j^{n+1} = T_j^{n+1} - T_j^n;\ \alpha^* = \alpha + 0.5uC\Delta x;\ \ L_x = \frac{1}{2\Delta x}\{-1,0,+1\};\ L_{xx} = \frac{1}{(\Delta x)^2}\{1,2,1\}.$$

For an estimation of the stability and accuracy of this numerical scheme, note that for the transport equation the Lax-Wendroff scheme can be interpreted as the FTCS scheme with a modified diffusion α^*. Then the stability condition becomes $0 \le C^2 \le 2S^* \le 1$, where $S^* = \alpha^* \Delta t \big/ (\Delta x)^2$. The required approximation accuracy $\cong O\left((\Delta t)^2, (\Delta x)^2\right)$ is achieved here for $\mathbf{R}_{cell} \le 2$ (in which case space oscillations are avoided).

6.2.2 Implicit schemes

Implicit numerical schemes for the diffusion equation is considered are the most effective in view of the absence of additional stability conditions. Here, we propose the most

effective numerical schemes for the solution of the transport equation. The finite-difference Crank-Nicholson numerical scheme approximates equation (6.12) by

$$\frac{\Delta T_j^{n+1}}{\Delta t} + \{uL_x - \alpha L_{xx}\} \times \left\{ \frac{T_j^n + T_j^{n+1}}{2} \right\} = 0, \tag{6.28}$$

which can be represented in algorithmic form as

$$\begin{aligned} &-(S + 0.5C)T_{j-1}^{n+1} + 2(1+S)T_j^{n+1} - (S - 0.5C)T_{j+1}^{n+1} = \\ &= (S + 0.5C)T_{j-1}^{n} + 2(1-S)T_j^{n} + (S - 0.5C)T_{j+1}^{n}. \end{aligned} \tag{6.29}$$

For the Crank-Nicholson scheme, the truncation error (T.E.) for the inaccuracy estimate is:

$$\text{T.E.} = \frac{u(\Delta x)^2}{6}\left(1 + 0.5C^2\right) T_{xxx} - \frac{\alpha(\Delta x)^2}{12}\left(1 + 3C^2\right) T_{xxxx} + \ldots \tag{6.30}$$

Equation (6.30) indicates that the algebraic equation (6.29) approximates the original equation (6.12) with accuracy $\cong O\left((\Delta t)^2, (\Delta x)^2\right)$, and the problem of artificial diffusion does not occur. The von Neumann stability analysis for this algorithm gives no additional restrictions either on C, or on S, although in order to eliminate oscillations the necessary accuracy is achieved by having $\mathbf{R}_{cell} = u\Delta x/\alpha \le 2$.

The three-layer fully implicit scheme (3LFI) can also used for the numerical solution of the transport equation (6.12):

$$\frac{3}{2}\frac{\Delta T_j^{n+1}}{\Delta t} - \frac{1}{2}\frac{\Delta T_j^{n}}{\Delta t} + \{uL_x - \alpha L_{xx}\} T_j^{n+1} = 0. \tag{6.31}$$

The truncation error for the 3LFI scheme is given by

$$\text{T.E.} = \frac{u(\Delta x)^2}{6}\left(1+2C^2\right)T_{xxx} - \frac{\alpha(\Delta x)^2}{12}\left(1+12C^2\right)T_{xxxx} + \ldots, \tag{6.32}$$

which also indicates that the algebraic equation (6.31) approximates the original equation (6.12) with second-order accuracy ($\cong O\left((\Delta t)^2,(\Delta x)^2\right)$) both in time and space variables, and that the problem of artificial diffusion does not arise. The von Neumann stability analysis yields no further restrictions either on C, or on S. To eliminate oscillations, the necessary solution accuracy is achieved by having $\mathbf{R}_{cell} = u\Delta x/\alpha \le 2$.

The analysis given above has indicated some of the positive properties of the algorithms of implicit numerical schemes for the transport equation, as regards their dissipative, as well as dispersive, characteristics. Next, we construct two new numerical schemes that also have accuracy $\cong O\left((\Delta t)^2,(\Delta x)^2\right)$, but which have much better dispersive characteristics. The first is a generalization of the Crank-Nicholson scheme with a mass operator. For the one-dimensional convection equation, this scheme is

$$M_x\left(\frac{T_j^{n+1}-T_j^n}{\Delta t}\right) + \frac{1}{2}\left(uL_x - \alpha L_{xx}\right)\left(T_j^n + T_j^{n+1}\right) = 0, \tag{6.33}$$

where $M_x \equiv \{\delta,(1-2\delta),\delta\}$ is the mass operator, and L_x, L_{xx} are finite-difference operators. The parameter δ is chosen so that the dispersive deficiency of the scheme is as small as possible. Obviously, scheme (6.33) results in a linear algebraic equation array with block tridiagonal matrix, which is easy to solve using the Thomas marching method. The von Neumann stability analysis reveals that the scheme is stable if $\delta \le 0.25$. As regards the long-wave characteristics of the numerical scheme, the modified equation is as follows:

$$\frac{\partial \overline{T}}{\partial t}+u\frac{\partial \overline{T}}{\partial x}-\alpha\frac{\partial^2 \overline{T}}{\partial x^2}+$$
$$+(\Delta x)^2\left[u\left(\frac{1}{6}+\frac{C^2}{12}-\delta\right)\frac{\partial^3 \overline{T}}{\partial x^3}-\alpha\left(\frac{1}{12}+\frac{C^2}{4}-\delta\right)\frac{\partial^4 \overline{T}}{\partial x^4}\right]+\ldots=0\ . \tag{6.34}$$

Formally, the numerical scheme (6.33) is second-order accurate, and it is obvious that the lowest dispersive term in the approximation deficiency can be suppressed by choosing $\delta = 1/6+C^2/12$. Furthermore, if $C^2 < 0.5$, the lowest dissipative term reveals positive dissipation in the approximation deficiency. However, for the optimal choice of the parameter δ, $C \le 1.0$ is required for scheme stability.

An alternative remedy for improving of the dispersive properties of the Crank-Nicholson scheme is to use the four-point upwind finite differencing scheme,

$$\frac{T_j^{n+1}-T_j^n}{\Delta t}+\frac{1}{2}\left(uL_x^{(4)}-\alpha L_{xx}\right)\left(T_j^n+T_j^{n+1}\right)=0\,, \tag{6.35}$$

where, for $u \ge 0$,

$$L_x^{(4)}T\equiv\frac{T_{j+1}-T_{j-1}}{2\Delta x}+\frac{q\left(T_{j-2}-3T_{j-1}+3T_j-T_{j-1}\right)}{3\Delta x}+O\left((\Delta x)^2\right). \tag{6.36}$$

Equation (6.36) represents the four-point convective operator introduced previously in conjunction with the analysis of the stationary convection-diffusion equation. Substitution of equation (6.36) into the numerical scheme (6.35) results in a linear algebraic equation array (LAEA) with a tetradiagonal matrix, which should be solved by the generalized Thomas algorithm.

The finite-difference scheme given by (6.35) is stable for $q \geq -3/\mathbf{R}_{cell}$. This does not constitute a practical restriction for the numerical algorithm, because only positive values of the parameter q are of interest here.

The following equation can be considered as an equivalent to the finite-difference equation (6.35):

$$\frac{\partial \overline{T}}{\partial t} + u\frac{\partial \overline{T}}{\partial x} - \alpha\frac{\partial^2 \overline{T}}{\partial x^2} + u(\Delta x)^2\left(\frac{(1-2q)}{6} + \frac{C^2}{12}\right)\frac{\partial^3 \overline{T}}{\partial x^3} - \\ -u\frac{(\Delta x)^3}{\mathbf{R}_{cell}}\left(\frac{1-2q\mathbf{R}_{cell}}{12} + \frac{C^2}{4}\right)\frac{\partial^4 \overline{T}}{\partial x^4} + \ldots = 0 \quad . \tag{6.37}$$

In equation (6.37), $u(\Delta x)^3/\mathbf{R}_{cell} = \alpha(\Delta x)^2$, using the definition of the grid Reynolds number, $\mathbf{R}_{cell} = u\Delta x/\alpha$, as given above.

For flows where $\mathbf{R}_{cell} >> 1$, the dispersive term provides the greatest contribution to the approximation deficiency, although this can be eliminated by choosing $q = 1/2 + C^2/4$. For this value of q and high grid Reynolds numbers $(\mathbf{R}_{cell})$, the dissipative term has the lowest order and introduces additional dissipation into the numerical finite-difference scheme.

The grid Reynolds number, $\mathbf{R}_{cell} = u\Delta x/\alpha$, can be interpreted as the local Reynolds number for a cell in a numerical grid in the case of a flow problem where α is the kinematic viscosity coefficient. If the heat transfer equation is solved, α is the heat diffusivity coefficient and $\mathbf{R}_{cell}$ is the local Péclet number calculated for a grid cell.

6.3 Exercises on hyperbolic PDEs and transport equations

1. Obtain the modified equation for the Lax scheme by solving the one-dimensional linear convection equation. Keep terms up to u_{xxx}.

2. Obtain the modified equation for the skip ("leap-frog") scheme by solving the one-dimensional linear convection equation. Keep terms up to u_{xxxxx}.

3. Consider the propagation of a truncated sinusoidal wave. Using the Lax scheme, solve the equation $\overline{T}_t + u\overline{T}_x = 0$, subject to the following initial and boundary conditions:

$$\overline{T}(x,0) = \begin{cases} \sin(10\pi x), & 0 \le x \le 0.1, \\ 0, & 0.1 < x \le 1.0 \end{cases} \qquad \overline{T}(0,t) = 0, \overline{T}(1,t) = 0.$$

Let $u = 0.1$ and Courant number $C = 0.8$. Use a 41-point grid and integrate for 40 time-steps. Compare your result with the exact solution of the problem.

4. Redo problem 3 using the explicit Lax-Wendroff second-order finite-difference scheme.

5. Redo problem 3 with the explicit second-order accurate skip ("leap-frog") scheme.

6. Redo problem 3 using the second-order accurate implicit time-centered finite-difference scheme.

7. Solve, using the FTCS scheme, the problem about the temperature front propagation for the equation $\overline{T}_t + u\overline{T}_x - \alpha\overline{T}_{xx} = 0$ with the following initial and boundary conditions:

$$\bar{T}(x,0) = \begin{cases} 1.0, & -2 \le x \le 0, \\ 0, & 0 < x \le 2, \end{cases} \qquad \bar{T}(0,t) = 0,\ \bar{T}(1,t) = 0 .$$

Let $u = 0.5,\ s = 0.25$ and Courant number $C = 0.25, 0.375, 0.5, 0.75$. Use a 21-point grid and integrate over 10 time-steps. Compare your results with the exact solution.

8. Redo problem 7 using the following explicit numerical schemes: (a) Lax-Wendroff scheme; (b) four-point upwind scheme.

9. Redo problem 7 with the implicit Crank-Nicholson numerical scheme.

10. Redo problem 7 using the modifications of the implicit Crank-Nicholson scheme with mass operator (6.33) and four-point Crank-Nicholson upwind scheme, (6.35) and (6.36). Compare with the results of the problem 9.

Optional tasks:

11. Introduce into to your computer program the central differencing scheme (6.4) and the upwind differencing scheme (6.5), and confirm the results shown in Fig. 6.1 for $\mathbf{R}_{cell} = 1,\ 2,\ 4$; $\Delta x = 0.05, 0.1, 0.2$; $u/\alpha = 20$. Compare with the exact solution.

12. Write computer programs to solve the linear convection-diffusion equation on the interval $0 < x < 1$ with homogeneous Dirichlet boundary conditions for: (a) the centred explicit scheme, (b) the Lax-Wendroff explicit scheme, and (c) the Crank-Nicholson scheme. The initial data is $f(x) = f_1(x)$ or $f_2(x)$ where $f_1(x) = 1$ for $0.4 \le x \le 0.6$ and zero elsewhere, while $f_2(x) = \sin(2\pi x)$. Let $a = -1$ and try both

$\nu = 1$ and $\nu = 0.00001$. Plot approximate solutions at $t = 0.4, 0.5, 0.6, 1.0$ and 2.0. In particular, try $N = 100$ for the number of grid points in x.

13. Analyze the dissipation and dispersion terms for the Lax-Wendroff scheme applied to the one-dimensional wave equation. Use the modified partial differential equation.

14. Write a computer program to solve the heat equation $\frac{\partial \overline{T}}{\partial t} + u\frac{\partial \overline{T}}{\partial x} - \alpha\frac{\partial^2 \overline{T}}{\partial x^2} = 0,$ subject to $\overline{T}(0,x) = \sin(\pi x)$, $\overline{T}(t,0) = \overline{T}(t,L) = 0$, using (a) the explicit scheme and (b) the implicit scheme. Plot approximate solutions at $t = 0.001, 0.06, 0.1, 0.9$ and 50.0. Experiment with the number of nodes, N (and hence the spatial step Δx), and time step Δt. Experiment with the values of the flow velocity u and the heat diffusivity α.

15. Obtain the modified equation for the Lax scheme and for the implicit Euler scheme by solving the wave equation. Keep terms up to u_{xxxx}.

16. Show that the numerical scheme obtained by solving the wave equation using the Rusanov method is equivalent to the following one-step finite-difference scheme:

$$u_j^{n+1} = u_j^n - \nu\left(\mu_x \overline{\delta}_x\right)\left(1 - \frac{\widehat{\delta}_x^2}{6}\right)u_j^n + \nu^2 \widehat{\delta}_x^2\left(\frac{1}{2} + \frac{\widehat{\delta}_x^2}{8}\right)u_j^n - \frac{\nu^3}{6}\left(\mu_x \overline{\delta}_x^3\right)u_j^n - \frac{\omega}{24}\widehat{\delta}_x^4.$$

17. Crawley proposed an explicit second-order finite-difference scheme for the solution of the wave equation in the form

$$u_j^{n+1} = u_j^n - \nu\left(\mu_x \overline{\delta}_x\right)u_j^n + \frac{\nu^2}{2}\left(\mu_x^2 \widehat{\delta}_x^2\right)u_j^n - \frac{1}{8}\nu^3\left(\mu_x \overline{\delta}_x^3\right)u_j^n.$$

Obtain the modified equation for this scheme. Keep terms up to u_{xxxx}.

18. Obtain modified equations for the Lax-Wendroff and Crank-Nicholson schemes for the one-dimensional transport equation.

19. Solve on a computer the first-order one-dimensional wave equation $u_t + u_x = 0$, subject to the initial condition $u(x,0) = \sin 2n\pi(x/L)$, $0 \le x \le L$ and periodic boundary conditions, by using: (a) the Lax scheme; (b) the Lax-Wendroff scheme. Let $n = 1,3$; $\nu = 1.0, 0.6, 0.3$. Use a 41-point grid with $\Delta x = 1$ and integrate up to $t = 18$. Compare the results obtained with the exact solution of this boundary problem (e.g. using the method of separated variables).

20. Repeat the previous problem for the following schemes: (a) McCormack scheme; (b) the Rusanov scheme for $\omega = 3$.

21. Solve on a computer the first-order one-dimensional wave equation $u_t + u_x = 0$ for the initial conditions,

$$u(x,0) = \begin{cases} 1, & x \le 10 \\ 0, & x > 10 \end{cases},$$

and Dirichlet boundary conditions using : (a) the upwind scheme; (b) the Crawley scheme (see problem 4). Let $n = 1,3$; $\nu = 1.0, 0.6, 0.3$. Use a 41-point grid with $\Delta x = 1$ and integrate up to $t = 18$. Compare the results obtained with the exact solution of the boundary-value problem.

22. Repeat the previous problem using the following numerical schemes: (a) two-step Lax-Wendroff scheme; (b) skip ("leap-frog") scheme.

7. The method of lines for PDEs

Both analytical as well as numerical methods are used to solve partial differential equations. The method of separated variables, the Fourier method, etc., which are studied in mathematical physics belong to the first group. An example of a numerical method is the finite-difference method. However, there is one group of methods, which uses approaches from both analytical and numerical methods. An example of this is the method of lines, which allows a reduction in the order of the equations. The main idea behind such methods is to obtain an approximate reduction of a boundary-value problem for a PDE to a solution of differential equations of lower order. For example, the solution of a two-dimensional PDE can be reduced to the solution of an ordinary differential equation array, or the solution of a three-dimensional problem can be reduced to a two-dimensional problem.

By the method of lines, in contrast to the finite-difference method, derivatives are approximated by means of grid functions for only some of the independent variables. Thus, in the numerical domain, the differential equation to be solved is also approximated by another differential equation, but with a reduced number of variables. This method is considered as a limiting case of the finite-difference method when the numerical mesh is refined for some of the variables. Here, we consider the method of lines for a linear second-order PDE, although it is useful also for non-linear PDEs of arbitrary order and for systems of partial differential equations.

7.1 The method of lines for the solution of parabolic PDEs

In the domain $G=\{(x,t): x\in[0,1], t\geq 0\}$, consider the following boundary-value problem,

$$\Lambda(u) \equiv L(u) - u_t = a(x,t)u_{xx} + b(x,t)u_x + c(x,t)u - u_t = f(x,t), \tag{7.1}$$

with the initial condition

$$u(x,0) = \varphi(x), \tag{7.2}$$

and the Dirichlet boundary conditions:

$$u(0,t) = \psi_0(t),\ u(1,t) = \psi_1(t). \tag{7.3}$$

Suppose that the coefficients in equation (7.1) and its functions, $a(x,y) \geq a > 0$, $b, c, f, \varphi, \psi_0, \psi_1$ satisfy all the conditions required for the existence and uniqueness of a solution to the boundary-value problem (7.1)-(7.3). We attempt to find a solution in the rectangle $\Pi = \{0 \leq x \leq 1,\ 0 \leq t \leq T < +\infty\}$. If x is treated as the continuous variable, the scheme is called transverse; otherwise, the longitudinal method of lines is obtained. First, we consider the transverse method of lines proposed by Rothe [45].

First, we generate the set of lines $t = t_n = n \cdot \Delta t$ $(n = 0,1,...,N;\ N = [T/\Delta t])$ with step-size $\Delta t > 0$, and find the approximate values of $u_n(x) \approx u(x,t_n)$ on those lines. For this, the time derivative in (7.1) is replaced by the left-hand finite-difference approximation,

$$u_t(x,t_n) \approx \frac{u(x,t_n) - u(x,t_{n-1})}{\Delta t}. \tag{7.4}$$

Then, taking into account boundary condition (7.2), a boundary-value problem for an equation array of N second-order ordinary differential equations (ODEs) is obtained on the segment $[0,1]$:

$$u_0(x) = \varphi(x), \tag{7.5}$$

$$\Lambda_n(u_n) \equiv a_n(x)\ u_n''(x) + b_n(x)\ u_n'(x) + \\ + c_n(x)u_n(x) - \frac{u_n(x) - u_{n-1}(x)}{\Delta t} = f_n(x), \tag{7.6}$$

$$u_n(0) = \psi_0(t_n),\ u_n(1) = \psi_1(t_n),\ n = \overline{1,N}. \tag{7.7}$$

The solution of the problem is known from (7.5) on the line $t = t_0$. Therefore the functions $u_n(x)$ are found on the lines $t = t_n$, $n = \overline{1,N}$ by successively solving one second-order ODE, (7.6), with boundary conditions (7.7) for each step n.

Remarks:

- *Because the method of lines can be considered as a limiting variant of the finite-difference method (FDM) as $\Delta x \to 0$, the scheme (7.5)-(7.7) corresponds to an implicit FDM scheme and is stable for arbitrary time-step, Δt. Meanwhile, the use of a right-hand difference instead of (7.4) leads to an explicit FDM, which converges only if $\Delta t \le O\left((\Delta x)^2\right)$, and is hardly applicable in practice as $\Delta x \to 0$.*

- *The numerical scheme constructed for the method of lines reveals the geometrical content of the method clearly enough. From the geometrical point of view, in a multi-dimensional case, the method is often called the method of planes or hyperplanes.*

The method of lines appears to be more precise than the finite-difference method because the replacement of a given problem by its approximation is much more accurate; however, the solution of the approximate equations requires greater computation time. Nevertheless, if the coefficients of equation (7.1) do not depend on x, then by using the method of lines one can get the equation array of second-order ODEs with constant coefficients, which can be solved analytically.

The inaccuracy associated with the Rothe method of lines can be computed by using Taylor series:

$$u(x,t_{n-1}) = u(x,t_n) - \Delta t u_t(x,t_n) + \frac{(\Delta t)^2}{2} u_{tt}(x,t_n - \theta_n \tau) \ ,$$
$$0 < \theta_n < 1 \ , \ n = \overline{1,N},$$

from which the inaccuracy can be calculated as

$$r_n(x) = u_t(x,t_n) - \frac{u(x,t_n) - u(x,t_{n-1})}{\Delta t} \approx O(\Delta t). \tag{7.8}$$

Evidently, the required accuracy cannot be achieved by the Rothe scheme. The order of accuracy can be increased in the following way: the time derivative at $t = t_n$ is replaced by a left-hand finite-difference, and for $t = t_{n-1}$ it is replaced by a right-hand finite-difference. Then the numerical scheme by the method of lines has the form:

$$u_0(x) = \varphi(x) \ ,$$

$$\frac{L(u_n) + L(u_{n-1})}{2} - \frac{u_n(x) - u_{n-1}(x)}{\Delta t} = \frac{f_n(x) + f_{n-1}(x)}{2}, \tag{7.9}$$

$$u_n(0) = \psi_0(t_n), \ u_n(1) = \psi_1(t_n), \ n = 1,2,...,N \ .$$

Here, $L(u_i) \equiv a_i(x) u_i''(x) + b_i(x) u_i'(x) + c_i(x) u_i(x), \ i = \overline{0,N}$. Scheme (7.9) is more precise than the Rothe scheme and has discrepancy $\approx O\left((\Delta t)^2\right)$.

Now consider the longitudinal variant of the method of lines for the boundary-value problem (7.1)-(7.3). Let us generate the set of lines $x = x_j = j \cdot \Delta x \ (j = 0,1,...,M)$ with a step $\Delta x > 0$ and compute the approximate functions $u_j(t) \approx u(x_j,t)$ on these lines. For this purpose, the space derivatives in equation (7.1) are replaced by the following symmetrical finite-difference approximations:

$$u_x\left(x_j,t\right)\approx\frac{u\left(x_{j+1},t\right)-u\left(x_{j-1},t\right)}{2\Delta x},$$
$$u_{xx}\left(x_j,t\right)\approx\frac{u\left(x_{j+1},t\right)-2u\left(x_j,t\right)+\ u\left(x_{j-1},t\right)}{(\Delta x)^2}.\tag{7.10}$$

The scheme of the method of lines for the longitudinal case has the form:

$$u_j'(t)=a_j(t)\frac{u\left(x_{j+1},t\right)-2u\left(x_j,t\right)\ +\ u\left(x_{j-1},t\right)}{(\Delta x)^2}\ +$$
$$+b_j(t)\frac{u\left(x_{j+1},t\right)-u\left(x_{j-1},t\right)}{2\Delta x}+c_j(t)u_j(t)+f_j(t)\ ,\ t>0\ ,\tag{7.11}$$

$$u_0(t)=\psi_o(t),\quad u_M(t)=\psi_1(t),\ t\ \geq\ 0\tag{7.12}$$

$$u_j(0)=\varphi\left(x_j\right),\ j\ =\ \overline{1,M-1}.\tag{7.13}$$

Thus the computation of the approximate functions $u_j(t)$ for the solution $u(x,t)$ of the boundary-value problem (7.1)-(7.3) on the lines $x=x_j\ \ (j=1,...,M-1)$ is reduced to a solution of the Cauchy problem for an equation array of $M-1$ linear first-order ODEs. The accuracy of approximation is $\approx\ O\left((\Delta x)^2\right)$.

Note that for solution of the Cauchy problem (7.11)-(7.13), numerical Runge-Kutta schemes having increased order of accuracy in time are used; consequently, an accuracy of $\approx\ O\left((\Delta x)^2\right)$ in the spatial approximation may be good enough. A similar strategy is used to increase the spatial approximation accuracy for longitudinal schemes; most of these are considered in monographs [20, 45, 57].

7.2 The method of lines for elliptic equations

Supposed that there exists a unique solution in the rectangle $\Pi = \{0 \le x \le X,\ 0 \le y \le Y\}$ for the boundary-value problem

$$\Lambda(u) \equiv u_{yy} + a(x,y)u_{xx} + b(x,y)u_x + c(x,y)u = f(x,y), \tag{7.14}$$

with the Dirichlet boundary conditions:

$$\left.\begin{aligned} u(x,0) = \varphi_0(x), \quad u(x,Y) = \varphi_1(x) \\ u(0,y) = \psi_0(y), \quad u(X,y) = \psi_1(y) \end{aligned}\right\}. \tag{7.15}$$

Assume that the conditions, $a(x,y) \ge a > 0,\ c(x,y) \le -c < 0$ are satisfied everywhere in Π. The substantial difference between transverse and longitudinal methods of lines disappears in the case of an elliptic equation. Divide Π into N bands by the lines $y = y_n = n\Delta y,\ n = \overline{1, N-1}$ with step-size $\Delta y = Y/N$. If u_{yy} is replaced by a finite-difference approximation on each of these bands, then a boundary-value problem is obtained that approximates the original boundary-value problem with Dirichlet boundary conditions:

$$u_0(x) = \varphi_0(x),\ u_N(x) = \varphi_1(x), \tag{7.16}$$

$$\begin{aligned} &\Lambda_n(u_n) \equiv \frac{u_{n+1}(x) - 2u_n(x) + u_{n-1}(x)}{(\Delta y)^2} + \\ &+a_n(x)\ u_n''(x) + b_n(x)\ u_n'(x) + c_n(x)\ u_n(x) = f_n(x), \end{aligned} \tag{7.17}$$

$$u_n(0) = \psi_0(y_n),\ u_n(X) = \psi_1(y_n),\ n = \overline{1, N-1}. \tag{7.18}$$

Thus, the solution of the boundary-value problem stated in (7.14)-(7.15) has been reduced to the solution of $N-1$ Dirichlet problems for linear second-order ODEs. The approximation inaccuracy is

$$r_n(x) = \frac{(\Delta y)^2}{12}\frac{\partial^4 u}{\partial y^4} \approx O\left((\Delta y)^2\right). \tag{7.19}$$

Evidently, it is possible to increase the order of approximation by using the approaches considered above, as well as by other ones proposed in the monograph by Krylov [45].

7.3 Solution of hyperbolic equations by the method of lines

Now consider the boundary-value problem for a hyperbolic equation in the rectangle $\Pi = \{0 \le x \le 1;\ 0 \le t \le T < +\infty\}$,

$$u_{tt} + g(x,t)u_t = a(x,t)u_{xx} + b(x,t)u_x + c(x,t)u + f(x,t), \tag{7.20}$$

with the initial conditions

$$u(x,0) = \varphi_0(x),\ u_t(x,0) = \varphi_1(x), \tag{7.21}$$

and Dirichlet boundary conditions:

$$u(0,t) = \psi_0(t),\ u(1,t) = \psi_1(t). \tag{7.22}$$

Suppose that $a(x,t) \ge a > 0$ and that there exists a unique solution to the boundary-value problem (7.20)-(7.22). An approximate solution is found on the lines

$$t = t_n = n\Delta t\ ,\ n = \overline{1,N};\ N = [T/\Delta t],\ \Delta t > 0.$$

The spatial second-order derivative is approximated on the line $t = t_{n+1},\ n = \overline{1,N-1}$ by central differences, according to

$$u_{tt}\left(x,t_{n+1}\right)=\frac{u\left(x,t_{n+1}\right)-2u\left(x,t_n\right)+u\left(x,t_{n-1}\right)}{(\Delta t)^2}+\Delta t\ \ u_{ttt}\left(x,t_n\right)+O\left((\Delta t)^2\right), \tag{7.23}$$

and the first-order time derivative is approximated by the left-hand difference

$$u_t\left(x,t_{n+1}\right)=\frac{u\left(x,t_{n+1}\right)-u\left(x,t_n\right)}{\Delta t}+\frac{\Delta t}{2}u_{tt}\left(x,\tau_n\right). \tag{7.24}$$

Then, on the line $t=0$, the following right-hand difference is used for the approximation of the initial condition:

$$u_t\left(x,0\right)\approx\frac{u\left(x,\Delta t\right)-u\left(x,0\right)}{\Delta t}. \tag{7.25}$$

Thus, the computational algorithm for the approximate functions $u_n\left(x\right)\approx u\left(x,\ t_n\right),\ n=\overline{0,N}$ is the following:

$$u_0\left(x\right)=\varphi_0\left(x\right),\ \frac{u_1\left(x\right)-u_0\left(x\right)}{\Delta t}=\varphi_1\left(x\right); \tag{7.26}$$

$$\begin{aligned}&\frac{u_{n+1}\left(x\right)-2\ \ u_n\left(x\right)-u_{n-1}\left(x\right)}{\left(\Delta t\right)^2}+g_{n+1}\left(x\right)\frac{u_{n+1}\left(x\right)-u_n\left(x\right)}{\Delta t}=\\&=a_{n+1}\left(x\right)\ \ u''_{n+1}\left(x\right)+b_{n+1}\left(x\right)u'_{n+1}\left(x\right)+c_{n+1}\left(x\right)u_{n+1}\left(x\right)+f_{n+1}\left(x\right),\\&u_{n+1}\left(0\right)=\psi_0\left(t_{n+1}\right),\ u_{n+1}\left(1\right)=\psi_1\left(t_{n+1}\right),\ n=\overline{1,N-1}.\end{aligned} \tag{7.27}$$

The solution of the differential-difference equation array (7.26), (7.27) is split into the following two stages:

1. $u_1\left(x\right)$ is computed by formula (7.26) on the line $t=t_1$.
2. The solutions $u_2(x),...,u_N\left(x\right)$ are calculated by equations (7.27) successively, step by step, subject to the boundary conditions.

The common approximation accuracy for the equations (7.26), (7.27) is $\approx O\left(\Delta t^2\right)$

$u_t\left(x,t_n\right) \approx \left\{3u\left(x,t_{n+1}\right) - 4u\left(x,t_n\right) + u\left(x,t_{n-1}\right)\right\} \Big/ \left\{2\Delta t\right\}$

$u_j''(t) + g_j(t)u_j'(t) = a_j(t)\dfrac{u\left(x_{j+1},t\right) - 2u\left(x_j,t\right) + u\left(x_{j-1},t\right)}{(\Delta x)^2} + \approx O\left(\Delta t\right)$. Using a second-

$+b_j(t)\dfrac{u\left(x_{j+1},t\right) - u\left(x_{j-1},t\right)}{2\ \Delta x} + c_j(t)\,u_j(t) + f_j(t)\ ,\ t > 0$

order approximation for the time derivative in (7.25),

$$u_t\left(x,t_n\right) \approx \frac{3u\left(x,t_{n+1}\right) - 4u\left(x,t_n\right) + u\left(x,t_{n-1}\right)}{2\Delta t}, \tag{7.28}$$

and a symmetry formula to approximate boundary condition (7.26) and eliminate the fictitious value in (7.27) for $n = 0$, one can get a common accuracy of the approximation which is not less than $\approx O\left((\Delta t)^2\right)$.

Applying the method of lines for the solution of hyperbolic equations, the longitudinal scheme can be also used, and this has the form:

$$\begin{aligned} &u_j''(t) + g_j(t)u_j'(t) = a_j(t)\frac{u\left(x_{j+1},t\right) - 2u\left(x_j,t\right) + u\left(x_{j-1},t\right)}{(\Delta x)^2} + \\ &+ b_j(t)\frac{u\left(x_{j+1},t\right) - u\left(x_{j-1},t\right)}{2\Delta x} + c_j(t)u_j(t) + f_j(t), \quad t > 0\ , \end{aligned} \tag{7.29}$$

$$u_0(t) = \psi_o(t),\ u_M(t) = \psi_1(t)\ ,\ t \geq 0\ , \tag{7.30}$$

$$u_j(0) = \varphi_0\left(x_j\right),\ u_j'(0) = \varphi_1\left(x_j\right),\ j = \overline{1, M-1}. \tag{7.31}$$

Thus the calculation of approximate functions $u_j(t)$ for the solution $u(x,t)$ of the boundary-value problem (7.20)-(7.22) on the lines $x = x_j$ $(j = 1,...,M-1)$ is reduced to the solution of a Cauchy problem for a system of $M-1$ linear second-order ODEs. The approximation accuracy is $\approx O\left((\Delta x)^2\right)$.

7.4 Exercises on the method of lines

1. Solve the normalized linear diffusion equation $u_t = u_{xx}$ ($0 \le x \le 1,\ t \ge t_0$) using the method of lines, with the following initial condition: $t_0 = 0,\ u(x,t_0) = u_0 = 1$ and boundary conditions: $u_x(1,t) = 0$, and $u(0,t) = 0$, which are abruptly changed from u_0 to $u_1 = 0.1$. Integrate by time till $t_{end} = 0.09$.

2. Solve the non-linear Burgers equation $u_t + \nu(x) u_x = \left[D(x) u_x \right]_x$ ($0 \le x \le 1,\ t \ge t_0$), using the method of lines, where

$$D(x) = \begin{cases} 5, & 0 \le x \le 0.5; \\ 1, & 0.5 \le x \le 1; \end{cases} \qquad \nu(x) = \begin{cases} 1000, & 0 \le x \le 0.5; \\ 1, & 0.5 \le x \le 1. \end{cases}$$

   The initial and boundary conditions are, respectively:

$$u(x,0) = \begin{cases} 1, & x = 0; \\ 0, & x > 0. \end{cases} \qquad u(0,t) = 1,\ u(1,t) = 0 \ .$$

   Obtain the graph of the function $u=u(x,t_{end})$.

3. Redo the task 1 by the initial condition $u(x,0) = 1 + \cos\left[(2n-1)\pi x \right]$ $n>1$ and Neumann boundary condition $u_x(0,t) = 0,\ u_x(1,t) = 0$.

4. Solve the normalized hyperbolic linear equation $u_{tt} = u_{xx}$ modelling the oscillating string with the fastened edges, by the following initial and boundary conditions, respectively:

$$u(x,0) = \sin(\pi x), \quad u_t(x,0) = 0; \qquad u_x(0,t) = 0,\ u_x(1,t) = 0.$$

5. Redo the task 2 from the chapter 3 using the method of lines. Compare with the exact analytical solution.

6. Solve the task 5 from the chapter 3 using the method of lines. Compare the solution obtained with the exact analytical solution.

7. Solve the task 3 from the chapter 4 using the method of lines. Compare the solution obtained with the exact analytical solution.

8. Solve the task 7 from the chapter 4 using the method of lines. Compare the solution obtained with the exact analytical solution.

Optional tasks:

9. Solve with the method of lines the Poisson equation $u_{xx} + u_{yy} = x + y$ in a square 1x1, with the Dirichlet boundary conditions:

$$u(0,x) = u(1,x) = x^2 + xy; \qquad u(y,0) = u(y,1) = y^2 + xy,.$$

10. Solve task 5 with the Neumann boundary conditions:

$$u_x(0,x) = u_x(1,x) = 2x + y;$$
$$u_y(y,0) = u_y(y,1) = 2y + x$$

.

References used and recommended

1. Amsden A.A., Harlow F.H. *The SMAC Method: A Numerical Technique for Calculating Incompressible Fluid Flows*. – Los Alamos, New Mexico.- Los Alamos Scientific Laboratory Report LA-4370.- 1970.

2. Anderson, D.A., Tannehill J.C., Pletcher R.H. *Computational Fluid Mechanics and Heat Transfer*: McGraw-Hill.- New York.- 1984.

3. Anderson, J.D., Jr. Computational Fluid Dynamics: the Basics with Applications: McGraw-Hill.- New York.- 1995.

4. Bakhvalov N.S., Zhidkov N.P., Kobel'kov G.M. *Numerical methods*.- Moscow: Nauka.- 1987.-600 p. (in Russian).

5. Baldwin B.S., Lomax H. *Thin layer Approximation and Algebraic Model for Separated Turbulent Flows//* AIAA Paper 78-257.- Huntsville. Alabama.- 1978.

6. Barth T. J. *Computational Fluid Dynamics*.- Lecture Series 1994-05.- March 21-25.- NASA Ames Research Center, Moffet Field, Ca., USA.- Von Karman Institute for Fluid Dynamics.- 1994.

7. Beam R.M., Warming R.F. *An Implicit Factored Scheme for the Compressible Navier-Stokes Equations//* AIAA Journal.- 1978.- V.16.- No. 4.- P. 393-402.

8. Belov I.A., Isaev S.A., Korobkov V.A. *The Problems and Numerical Methods for Separated Imcompressible Inviscid Flows.*- Leningrad: Sudostroenie.- 1989.- 254 p. (in Russian).

9. Briley W.R. *A Numerical Study of Laminar Separation Bubbles using the Navier-Stokes Equations*.- United Aircraft Research Laboratories.- Report J110614-1.- East Hartford, Connecticut.- 1970.

10. Briley W.R., McDonald H. *Analysis and Computation of Viscous Subsonic Primary and Secondary Flows//* AIAA Paper 79-1453.- Williamsburg, Virginia.- 1979.

11. Burkivs'ka V.L., Voitsekhivs'kiy S.O., Gavrylyuk I.P., etc. *Numerical Methods. Practical Work on Computer*.- Kyiv.: Vyshcha shkola.- 1995.- 303 pp. (in Ukrainian).

12. Cebeci T., Bradshaw P. *Momentum Transfer in Boundary Layers*.- 1977.- New-York: McGraw-Hill.- 412pp.

13. Cebeci T., Bradshaw P. *Solutions Manual and Computer Programs for Physical and Computational Aspects of Convective Heat Transfer*.- Springer-Verlag.- 1989.

14. Chorin A.J. *A Numerical Method for Solving Incompressible Viscous Flow Problems*// J. Comput. Phys., 1967.- V.2.- P. 12-26.

15. Cushman J.H. *Continuous Families of Lax-Wendroff Schemes*/ Int. J. of Numerical Methods Engineering.- 1989.- V. 17.- P. 975.

16. Douglas J., Rackford H.H. *On the Numerical Solution of Heat Conduction Problems in Two and Three Space Variables*// Trans. Am. Math. Soc.- 1956.- V. 82.- P. 4231-4239.

17. Eberle A., Rizzi A., Hirschel E.H. *Numerical Solutions of the Euler Equations for Steady Flow Problems*// Notes on Numerical Fluid Mechanics.- Braunschweig; Wiesbaden : Vieweg. – 1992.- V. 34.

18. *Euler and Navier-Stokes Solvers Using Multi-Dimensional Upwind Schemes and Multigrid Acceleration/* Notes on Numerical Fluid Mechanics Ed. by Deconinck H., Koren B.- European Community Research in Aeronautics.- Vieweg Verlag.- Braunschweig.- 1997.- V. 57.- 569pp.

19. *Finite Difference Techniques For Vectorized Fluid Dynamic Calculations*/ Ed. Book D.L.- Springer Verlag.- 1981.

20. Fletcher C.A.J. *Computational Techniques for Fluid Dynamics*. Vol. I: Fundamental and General Techniques. Vol. II: Scecific Techniques for Different Flow Categories. Second Edition.- Springer Verlag.- Berlin. - 1991.

21. Forsythe G.E., Malcolm M.A., and Moler C.B. *Computer Methods for Mathematical Computations.* Prentice-Hall. - Englewood Cliffs.- 1977.

22. Fromm J. E., *Numerical Solutions of the Nonlinear Equations for a Heated Fluid Layer*// Phys. Fluids.- 1965.- V. 8.

23. Fromm J. E., *The Time Dependent Flow of an Incompressible Viscous Fluid*. In: Methods of Computational Physics. Vol. 3.- Ed. by Alder B., Fernbach S., Rotenberg M.- New York: Academic Press.- 1964.

24. Godlewski E., Raviart P.A. *Hyperbolic Systems of Conservation Laws*.- Mathematiques et Applications.- Ellipses Publications.- 1991.

25. Golub G.H., Loan C.F. *Matrix Computations*.- 2nd ed.- The John Hopkins University Press.- 1989.

26. Gosman A.D., Spalding D. B. *The Prediction of Confined Three-dimensional Boundary Layers*/ Salford Symposium on Internal Flows.- Inst. Mech. Engrs.- London.- Paper 19.- 1971.

27. Henshaw W., Kreiss H.-O., Reyna L. *On the Smallest Scale for the Incompressible Navier-Stokes Equations*// Theoretical and Computational Fluid Dynamics.- 1989.- V. 1.- P. 1-32.

28. Henshaw W., Kreiss H.-O., Reyna L. *Smallest Scale Estimates for the Incompressible Navier-Stokes Equations*// Arch. Rational Mech. Anal.- 1990.- 112.- P. 21-44.

29. *High Resolution Upwind And TVD Methods For The Compressible Flow Equations*.- A von Karman Institute Book.- 1994.

30. Hirsch C. *Numerical Computation of Internal and External Flows*.- John Wiley & Sons.- 1990.- Vol. 1 and 2.

31. Hjertager B.H. *Computer Simulation of Turbulent Reactive Gas Dynamics*.- In: Modelling, Identification and Control.- 1985.- 5:4, P. 211-236.

32. Holt M. *Numerical Methods in Fluid Mechanics.*- 2nd ed.- Springer-Verlag.- 1984.

33. *Incompressible Computational Fluid Dynamics*/ Ed. by Gunzburger M.D. & Nicolaides R.A.- Cambridge University Press.- 1993.

34. Jameson A. *Numerical Methods in Fluid Dynamics*.- V. 1127 of Lecture Notes in Mathematics (ch. Transonic Flow Calculations for Aircraft, P.156): Springer-Verlag.- 1983.

35. Janenko N.N. *The Split-Step Method for the Solution of Multi-Dimensional Mathematical Physics Problems*.- Novosibirsk: Nauka.- 1967.- 195 pp. (in Russian).

36. Jensen V.G. *Viscous Flow Round a Sphere at Low Reynolds Number ($\leq$ 40)*// Proc. Roy. Soc. London.- 1959.- Ser. A.- V. 249.- P.346–366.

37. Kalion V.A. *Numerical Methods for Solution of Boundary Problems in Continuum Mechanics*.- Kyiv.: Kyiv University Computer Center.- 1999.- 106 p. (in Ukrainian).

38. Kitchens (Jr.) C.W., Sedney R., Gerber N. *The Role of the Zone of Dependence Concept in Tree-Dimensional Boundary-Layer Calculations//* Proc. AIAA 2nd Computational Fluid Dynamics Conference.- Hartford, Connecticut.- 1975.- P. 102–112.

39. Knupp P., Steinberg S. *Fundamentals of Grid Generation*.- CRC Press.- 1994.

40. Knuth D.E. *Literate Programming*. Chicago: University of Chicago Press.- Ill.- 1992.

41. Kotake S., Hijikata K. *Numerical Simulation of Heat Transfer and Fluid Flow on a Personal Computer*: Elsevier.- V. 3: Transport Processes in Engineering.- 1993.

42. Krause E. *Numerical Treatment of Boundary Layer Problems*.- AGARD LS-64.- Brussels.- NATO.- 1973.- P. 4.1-4.21.

43. Krause E., Schmidt W. *Euler Solutions as Limit of Infinite Reynolds Number for Separated Flows and Flows with Vortices//* Proc. of 8th Int. Conf. on Numerical Methods in Fluid Dynamics.- Lecture Notes in Physics.- Ed. by Krause E.- Aachen.- June 1982.: Springer-Verlag.- 1982.- V. 170.- P.468-473.

44. Kreiss H.O. & Lorenz J. *Initial-Boundary Value Problems and the Navier-Stokes Equations//* Pure and applied mathematics.- Academic Press.- 1989.- V. 136.

45. Krylov V.I., Bobkov V.V., Monastyrniy P.I. *Nunerical Methods*. Parts I and II.- Moscow: Nauka.- 1977.- 704 pp. (in Russian).

46. Li C.P. *Numerical Solution of Viscous Reacting Blunt Body Flows of a Multicomponent Mixture//* AIAA Paper 77-168.- Los Angeles, California.- 1973.

47. Li C.P. *A Numerical Study Separated Flows Induced by Shock-Wave/Boundary-Layer Interaction//* AIAA Paper 77-168.- Los Angeles, California.- 1977.

48. Lyashko I.I., Макarov V.L., Skorobogat'ko A.A. *The Methods of Calculation*.- Kyiv: Vyshcha shkola.- 1977.- 406 pp. (in Russian).

49. MacCormack R.W. *An Efficient Numerical Method for Solving the Time-Dependent Compressible Navier-Stokes Equations at High Reynolds Number*.- NASA TM X-73, 129.- 1976.

50. MacCormack R.W. *Numerical Solution of Compressible Viscous Flows at High Reynolds Number//* AIAA J.- 1982.- V. 20.- P. 1275.

51. MacCormack R.W. *Current Status of Numerical Solutions of the Navier-Stokes Equations//* AIAA paper 85-0032.- 1985.

52. MacCormak R.W., Baldwin B.S. *A Numerical Method for Solving the Navier-Stokes Equations with Application to Shock-Boundary Layer Interactions*.- AIAA Paper 75-1.-Pasadena.- California.- 1975.

53. Marchuk, G. I. *Methods of Numerical Mathematics*.- New York: Springer-Verlag.- 1975.- 316pp.

54. Minkowycz W.J., Sparrow E.M., Schneider G.E. and Pletcher R.H. *Handbook of Numerical Heat Transfer*.- Wiley Interscience: John Wiley & Sons, Inc.- 1988.- 1024pp.

55. Miyakoda K. *Contribution to the Numerical Weather Prediction – Computation with Finite Difference//* Japan J. Geophys.- 1962.- V.3.- P. 75–190.

56. *Numerical Methods for Advection-Diffusion Problems/* Ed. by Vreugdenhil C.B., Koren B.- Notes on Numerical Fluid Mechanics.- Vieweg, Braunschweig.-1993.- V.45.

57. Oran, E.S., Boris, J.P. *Numerical Simulation of Reactive Flow*. New York: Elsevier.- 1987.

58. *Parallel Computational Fluid Dynamics: Implementations and Results/* Ed. by Simon H.D.- The MIT Press.-1992.

59. Paskonov V.M., Polezhaev V.I., Chudov L.A. *Numerical Simulation of Heat- and Masstransfer Processes*.- Moscow: Nauka.- 1984.- 286 pp. (in Russian).

60. Patankar S.V., Spalding D.B. *A Calculation Procedure for Heat, Mass and Moment Transfer in Three-Dimensional Parabolic Flows//* Int. Journal of Heat and Mass Transfer.- 1972.- V.15, P. 1786-1806.

61. Patankar S.V. *Numerical Heat Transfer and Fluid Flow*.- McGraw-Hill.- 1981.

62. Peaceman D.W., Rackford H.H. *The Numerical Solution of Parabolic and Elliptic Differential Equations//* J. Soc. Ind. Appl. Math.- 1955.- V.3.- P. 28-41.

63. Peyret R. and Taylor T. D.. *Computational Methods for Fluid Flow.* Springer-Verlag, 1990.- 359pp.

64. Randall J. *Numerical Methods for Conservation Laws*.- LeVeque: Birkhauser Verlag.- 1992.

65. Roache P.J. *Computational Fluid Dynamics*. Hermosa. Publisher.- Albuquerque.- 1977.

66. Roache P.J. Short communications// Int. J.Num. Methods in Fluids.- 1988.- V.8.- P. 1459-63.

67. Roache P.J. *Verification and Validation in Computational Science and Engineering*.-Hermosa Publishers, Albuquerque.- New Mexico.- 1998.

68. Rubin S.G. *A Review of Marching Procedures for Parabolized Navie-Stokes Equations//* Symposium on Numerical and Physical Aspects of Aerodynamic Flows.- New York: Springer-Verlag.- 1981.- P. 171-186.

69. Rubin S.G., Lin T. C. *Numerical Methods for Two- and Three-dimensional Viscous Flow Problems: Application to Hypersonic Leading Edge Equations//* Polytechnic Institute of Brooklyn.- New York: Farmingdale.- PIBAL Report 71-8.- 1971.

70. Samarskii A.A. *The Theory of Finite Difference Schemes*.- Moscow: Nauka.- 1983.- 616pp.

71. Samarskii A.A. *Introduction to Numerical Methods*.- Moscow: Nauka.- 1987.- 287pp.

72. Samarskii A.A., Nikolaev E.S. *Numerical Methods for Grid Equations.* Birkhauser Boston.- 1989.- 744pp.

73. Shang J.S. *Implicit-explicit method for Navier-Stokes equations//* AIAA Journal.- 1978.- V. 16.- No. 5, P. 102–109.

74. Shaw C.T. *Using Computational Fluid Dynamics*.- Prentice Hall.- 1992.

75. Srinivas K., Fletcher C.A. *A Solution Manual//* J. Computational Techniques for Fluid Dynamics: Springer Verlag.- 1992.

76. Steger J.L., Kutler P. *Implicit Finite-Difference Procedures for the Computation of Vortex Wakes*. – San Diego, California.- AIAA Paper 76-385.- 1976.

77. Steger J.L. *Implicit Finite-Differece Simulation of Flow about Arbitrary Two-Dimensional Geometries*// AIAA Journal.- 1978.- V. 16.- No. 7.

78. Tannehill J.C., Holst T.L. *Numerical Computation of Two-Dimensional Viscous Blunt Body Flows with an Impinging Shock*// AIAA Journal.- 1976.- V.14, No.2, P. 94–103.

79. Thompson J.F., Warsi Z.U.A., Mastin C.W. *Numerical Grid Generation. Foundations and Applications*.- North Holland.- 1985.

80. Tolstykh A.I. *High Accuracy Non-Centered Compact Difference Schemes for Fluid Dynamics Applications*: World Scientific.- 1994.

81. Turek S. *Efficient Solvers for Incompressible Flow Problems. An Algorithmic and Computational Approach.* (with CD-ROM).- University of Heidelberg.- Germany.- 1999.- XV.- 352 pp.

82. Vigneron Y.C., Rakich J.V., Tannehill J.C. *Calculation of Supersonic Viscous Flow over Delta Wings with Sharp Subsonic Leading Edges*.- AIAA Paper 78–1137.-Seattle, Washington.- 1978.

83. Volkov E.A. *Numerical Methods*.- Moscow: Nauka.- 1987.- 248 pp. (in Russian).

84. Voropaev G.A., Ptukha Yu.A. *Modelling of Turbulent Complex Flows*.- Kyiv: Naukova dumka.- 1991.- 167 pp. (In Russian).

85. Wang K. C. *Three-dimensional Laminar Boundary Layer over a Body of Revolution at Incidence*. Part VI: General Methods and Results of the Case at High Incidence.- Martin Marietta Laboratories.- Baltimore.- Maryland.- 1973.- MML TR 73–02c.

86. Warming R.F., Beam R.M. *On the Construction and Spplication of Implicit Factored Schemes for Conservation Laws* // SIAM AMS Proceedings.- 1978.- V.11.- P. 159–179.

87. Warsi Z.U.A. *Fluid Dynamics: Theoretical and Computational Approaches*.- CRC Press.- 1993.

88. Wei Shyy. *Computational Modeling for Fluid Flow and Interfacial Transport*.- Elsevier.- 1993.

89. Wendt J.F. (Ed.), Anderson J.D., Degrez G., Dick E., and Grundmann R. *Computational Fluid Dynamics-An Introduction*.- A von Karman Institute Book.- Springer-Verlag.- 1992.

90. Wesseling P. *An Introduction to Multigrid Methods*.- John Wiley & Sons.- 1992.

91. Williams F.A. *Combustion Theory*. Benjamin/Cummins, Menlo Park, Cal.- 1985.

92. Xiaojun Chen, Zuhair Nashed, Liqun Qi. *Convergence of Newton's Method for Singular Smooth and Nonsmooth Equations Using Adaptive Outer Inverses* // SIAM Journal on Optimization.- 1997.- V. 7.- N 2.- P. 445-462.

93. Young D. *Iterative methods for solving partial difference equations of elliptic type*// Trans. Amer. Math. Soc.- 1954.- V.- 76.- No.92-11.

94. Young D.M., *Iterative Solution of Large Linear Systems.*- New York: Academic Press.- 1971.

Key and solutions for exercises

Chapter 1

1. Considering the outgoing equation (1.4), let us perform the transformation of independent variables: $(x,y)\to(\xi,\eta)$. This transformation is non-degenerated if and only if the Jacobian of transformation is nonzero

$$J=\frac{\partial(\xi,\eta)}{\partial(x,y)}=\xi_x\eta_y-\xi_y\eta_x\ . \tag{K-1.1}$$

With using the chain rule for partial differentiation, the partial derivatives become:

$$\frac{\partial u}{\partial x}=\xi_x\frac{\partial u}{\partial \xi}+\eta_x\frac{\partial u}{\partial \eta}, \tag{K-1.2}$$

$$\frac{\partial u}{\partial y}=\xi_y\frac{\partial u}{\partial \xi}+\eta_y\frac{\partial u}{\partial \eta}, \tag{K-1.3}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2}=\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right)=\xi_x^2\frac{\partial^2 u}{\partial \xi^2}+2\xi_x\eta_x\frac{\partial^2 u}{\partial \xi\partial\eta}+\eta_x^2\frac{\partial^2 u}{\partial \eta^2}\\ +\xi_{xx}\frac{\partial u}{\partial \xi}+\eta_{xx}\frac{\partial u}{\partial \eta}\end{aligned} \tag{K-1.4}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial y^2}=\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}\right)=\xi_y^2\frac{\partial^2 u}{\partial \xi^2}+2\xi_y\eta_y\frac{\partial^2 u}{\partial \xi\partial\eta}+\eta_y^2\frac{\partial^2 u}{\partial \eta^2},\\ +\xi_{yy}\frac{\partial u}{\partial \xi}+\eta_{yy}\frac{\partial u}{\partial \eta}\end{aligned} \tag{K-1.5}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial x\partial y}=\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}\right)=\xi_x\xi_y\frac{\partial^2 u}{\partial \xi^2}+\left(\xi_x\eta_y+\xi_y\eta_x\right)\frac{\partial^2 u}{\partial \xi\partial\eta}.\\ +\eta_x\eta_y\frac{\partial^2 u}{\partial \eta^2}+\xi_{xy}\frac{\partial u}{\partial \xi}+\eta_{xy}\frac{\partial u}{\partial \eta}\end{aligned} \tag{K-1.6}$$

Substituting (K-1.2)-(K-1.6) in the equation (1.4) yields

$$A u_{\xi\xi}+2B u_{\xi\eta}+C u_{\eta\eta}+\ldots=F\ , \tag{K-1.7}$$

where are:

$$\begin{aligned}&A=a\xi_x^2+2b\xi_x\xi_y+c\xi_y^2;\\ &B=a\xi_x\xi_y+b\left(\xi_x\eta_y+\xi_y\eta_x\right)+c\xi_y\eta_y;\\ &C=a\eta_x^2+2b\eta_x\eta_y+c\eta_y^2.\end{aligned} \tag{K-1.8}$$

Now compute the determinant (K-1.7)

$$D = B^2 - AC = \left(b^2 - ac\right)\left[\xi_x \eta_y - \xi_y \eta_x\right]^2 = \delta \cdot J^2 \cdot \tag{K-1.9}$$

The Jacobian is always positive, thus, the sign of the determinant δ for the previous equation determines the sign of the determinant D for the new equation. Therefore the type of the equation remains the same.

2. Consider the example

$$u_{xx} - u_{yy} = h_4\left(u_x\ ,\ u_y\ ,\ u\ ,\ x\ ,\ y\right) \cdot \tag{K-1.10}$$

Here are $a=1,\ в=0,\ c=-1$. The coordinate transformation $(x,y) \rightarrow (\xi, \eta)$ is chosen to transform (K-1.10) to the form (1.7). Comparing (1.7) and (K-1.7)-(K-1.8), one can get two differential equations to obtain the transformation required

$$A=0,\ C=0 \tag{K-1.11}$$

Or

$$\begin{cases} \xi_x^2 - \xi_y^2 = 0, \\ \eta_x^2 - \eta_y^2 = 0, \end{cases} \qquad \begin{cases} \left(\xi_x - \xi_y\right)\left(\xi_x + \xi_y\right) = 0, \\ \left(\eta_x - \eta_y\right)\left(\eta_x + \eta_y\right) = 0. \end{cases} \tag{K-1.12}$$

The equation array (K-1.12) is easy transformed to the following

$$\begin{cases} \dfrac{\xi_x}{\xi_y} = 1 \ \cap \ \dfrac{\xi_x}{\xi_y} = -1, \\ \dfrac{\eta_x}{\eta_y} = 1 \ \cap \ \dfrac{\eta_x}{\eta_y} = -1, \end{cases} \tag{K-1.13}$$

which has the solution on the characteristics

$$\frac{dy}{dx} = \pm 1 \cdot \tag{K-1.14}$$

Now integration of the equations (K-1.14) yields the required transformation

$$\begin{cases} x + y = \xi, \\ x - y = \eta. \end{cases} \tag{K-1.15}$$

which is non-degenerated due to $J=2$. Substitution of (K-1.15) in (K-1.10) results in

$$u_{\xi\eta} = h_3\left(u_\xi, u_\eta, u, \xi, \eta\right). \qquad \text{(K-1.16)}$$

3. For the equation (1.6) *a=1, в=0, c=0*. Write the determinant:

$$\delta(x, y) = b^2 - a \cdot c = 0 - 1 \cdot 0 = 0 .$$

The task has been done.

4. For this equation, *a=1, в=0, c=1*. Write the determinant:

$$\delta(x, y) = b^2 - a \cdot c = 0 - 1 \cdot 1 < 0 ,$$

where from follows directly that this is elliptic equation.

5. Here *a=1, в=-1/2, c=0*. Similar to the previous task:

$$\delta(x, y) = b^2 - a \cdot c = 1/4 - 1 \cdot 0 > 0 .$$

where from follows that this is hyperbolic equation.

6. *а/* by the Hellwig's[1] method applied for the variables *(t,x)* it is obtained

$$\underline{A} = \begin{Bmatrix} 1 & 0 \\ 0 & -1 \end{Bmatrix}, \quad \underline{B} = \begin{Bmatrix} 0 & 1 \\ 1 & 0 \end{Bmatrix}.$$

This results in

$$|\underline{A}| = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} = -1, \quad |\underline{B}| = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = -1,$$

$$|\underline{C}| = \begin{vmatrix} 1 & 1 \\ 0 & 0 \end{vmatrix} + \begin{vmatrix} 0 & 0 \\ -1 & 1 \end{vmatrix} = 0.$$

The determinant is $D = |\underline{C}|^2 - 4|\underline{A}| \cdot |\underline{B}| = 0\text{-}4(\text{-}1)(\text{-}1)\text{=-}4\text{<}0$. Thus, the equation is elliptic.

б/ Now by Hellwig's method applied for the variables *(t,y)* yields

[1] Hellwig G. *Partial Differential Equations.* Stuttgart: B.G.Teubner.- 1977.

$$|\underline{A}| = \begin{vmatrix} 1 & -1 \\ 0 & 0 \end{vmatrix} = 0, \quad |\underline{B}| = \begin{vmatrix} 0 & 0 \\ 1 & 1 \end{vmatrix} = 0,$$
$$|\underline{C}| = \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} + \begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix} = 1 - 1 = 0.$$

The determinant is $D = |\underline{C}|^2 - 4|\underline{A}| \cdot |\underline{B}| = 0\text{-}4 \cdot 0 \cdot 0 = 0$. The equation is parabolic.

7. For this equation $a=1$, $в=1/2$, $c=1$. The determinant is:

$$\delta(x, y) = b^2 - a \cdot c = 1/4 - 1 \cdot 1 < 0,$$

where from follows that the equation is elliptic.

8. For this equation $a=1$, $в=1$, $c=5$. The determinant is:

$$\delta(x, y) = b^2 - a \cdot c = 1 - 1 \cdot 5 < 0.$$

This is elliptic equation.

9. Here are: $a=1$, $в=3$, $c=9$. This is parabolic equation:

$$\delta(x, y) = b^2 - a \cdot c = 9 - 1 \cdot 9 = 0.$$

10. Here are: $a=1$, $в=1$, $c=1$. This is parabolic equation:

$$\delta(x, y) = b^2 - a \cdot c = 1 - 1 \cdot 1 = 0.$$

Chapter 3

1. By definition, the point on the grid chosen is considered as internal for the mesh chosen

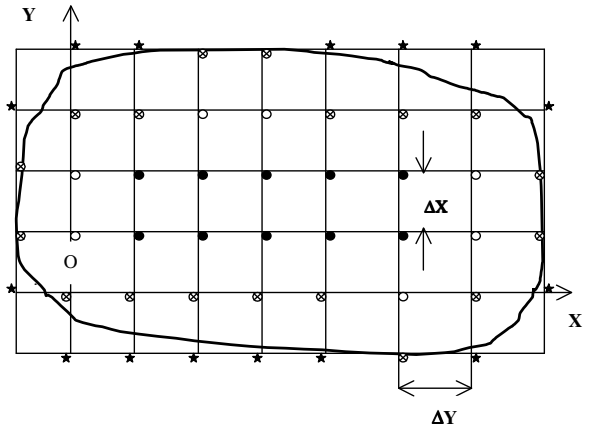


Fig. K- 3.1. Internal points

if this point and the surrounding points of the mesh are located inside the numerical domain or at its boundary. In accordance with this, the non-painted points in Fig. K-3.1 (Fig. 2.2) are internal for the "criss-cross" mesh (Fig. 2.5a). Similarly, for the "compact molecule" (Fig. 2.5b) the internal points are painted over points in Fig. K-3.1. The rest of the non-painted points belong to a set of the boundary points at the nine-point "compact molecule".

2. Using (2.11), (2.12), write FDE for the Laplace's equation $u_{xx} + u_{yy} = 0$:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0 \qquad \text{(K-3.1)}$$

Then apply Taylor series in the points $u_{i+1,j}; u_{i-1,j}; u_{i,j+1}; u_{i,j-1}$ by Δx, Δy, respectively:

$$u_{i+1,j} = u_{ij} + \Delta x \cdot u_x + \frac{\Delta x^2}{2} \cdot u_{xx} + \frac{\Delta x^3}{6} \cdot u_{xxx} + \\ + \frac{\Delta x^4}{24} \cdot u_{xxxx} + \frac{\Delta x^5}{120} \cdot u_{xxxxx} + \frac{\Delta x^6}{720} \cdot u_{xxxxxx} + \ldots \qquad \text{(K-3.2)}$$

$$u_{i-1,j} = u_{ij} - \Delta x \cdot u_x + \frac{\Delta x^2}{2} \cdot u_{xx} - \frac{\Delta x^3}{6} \cdot u_{xxx} + \\ + \frac{\Delta x^4}{24} \cdot u_{xxxx} - \frac{\Delta x^5}{120} \cdot u_{xxxxx} + \frac{\Delta x^6}{720} \cdot u_{xxxxxx} + \ldots \qquad \text{(K-3.3)}$$

$$u_{i,j+1} = u_{ij} + \Delta y \cdot u_y + \frac{\Delta y^2}{2} \cdot u_{yy} + \frac{\Delta y^3}{6} \cdot u_{yyy} + \\ + \frac{\Delta y^4}{24} \cdot u_{yyyy} + \frac{\Delta y^5}{120} \cdot u_{yyyyy} + \frac{\Delta y^6}{720} \cdot u_{yyyyyy} + \ldots \tag{K-3.4}$$

$$u_{i,j-1} = u_{ij} - \Delta y \cdot u_y + \frac{\Delta y^2}{2} \cdot u_{yy} - \frac{\Delta y^3}{6} \cdot u_{yyy} + \\ + \frac{\Delta y^4}{24} \cdot u_{yyyy} - \frac{\Delta y^5}{120} \cdot u_{yyyyy} + \frac{\Delta y^6}{720} \cdot u_{yyyyyy} + \ldots \tag{K-3.5}$$

Now substitute (K-3.2)- (K-3.5) into (K-3.1) and get the modified equation

$$u_{xx} + u_{yy} = -\frac{1}{12}\left(u_{xxxx}(\Delta x)^2 + u_{yyyy}(\Delta y)^2\right) + \ldots \tag{K-3.6}$$

In correspondence with (K-3.6) the approximation deficiency is $\approx O((\Delta x)^2, (\Delta y)^2)$.

3. Using (2.11), (2.12) and (2.14), derive FDE for PDE $u_{xx} + u_{xy} + u_{yy} = 0$:

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i+1,j+1} - u_{i-1,j+1} - u_{i+1,j-1} + u_{i-1,j-1}}{4\Delta x \Delta y} + \\ + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} = 0. \tag{K-3.7}$$

4. Use (K-3.2)-(K-3.5) together with Taylor series for $u_{i+1,j+1}; u_{i-1,j+1}; u_{i+1,j-1}; u_{i-1,j-1}$ by two variables simultaneously:

$$u_{i+1,j+1} = u_{ij} + \Delta x \cdot u_x + \Delta y \cdot u_y + \frac{\Delta x^2}{2} \cdot u_{xx} + \Delta x \Delta y \cdot u_{xy} + \\ + \frac{\Delta y^2}{2} \cdot u_{yy} + \frac{\Delta x^3}{6} \cdot u_{xxx} + \frac{\Delta x^2 \Delta y}{2} \cdot u_{xxy} + \frac{\Delta x \Delta y^2}{2} \cdot u_{xyy} + \\ + \frac{\Delta y^3}{6} \cdot u_{yyy} + \frac{\Delta x^4}{24} \cdot u_{xxxx} + \frac{\Delta x^3 \Delta y}{6} \cdot u_{xxxy} + \\ + \frac{\Delta x^2 \Delta y^2}{4} \cdot u_{xxyy} + \frac{\Delta x \Delta y^3}{6} \cdot u_{xyyy} + \frac{\Delta y^4}{24} \cdot u_{yyyy} + \ldots, \tag{K-3.8}$$

$$u_{i-1,j+1} = u_{ij} - \Delta x \cdot u_x + \Delta y \cdot u_y + \frac{\Delta x^2}{2} \cdot u_{xx} - \Delta x \Delta y \cdot u_{xy} + \\ + \frac{\Delta y^2}{2} \cdot u_{yy} - \frac{\Delta x^3}{6} \cdot u_{xxx} + \frac{\Delta x^2 \Delta y}{2} \cdot u_{xxy} - \frac{\Delta x \Delta y^2}{2} \cdot u_{xyy} + \\ + \frac{\Delta y^3}{6} \cdot u_{yyy} + \frac{\Delta x^4}{24} \cdot u_{xxxx} - \frac{\Delta x^3 \Delta y}{6} \cdot u_{xxxy} + \\ + \frac{\Delta x^2 \Delta y^2}{4} \cdot u_{xxyy} - \frac{\Delta x \Delta y^3}{6} \cdot u_{xyyy} + \frac{\Delta y^4}{24} \cdot u_{yyyy} + \ldots, \tag{K-3.9}$$

$$u_{i+1,j-1} = u_{ij} + \Delta x \cdot u_x - \Delta y \cdot u_y + \frac{\Delta x^2}{2} \cdot u_{xx} - \Delta x \Delta y \cdot u_{xy} + \\ + \frac{\Delta y^2}{2} \cdot u_{yy} + \frac{\Delta x^3}{6} \cdot u_{xxx} - \frac{\Delta x^2 \Delta y}{2} \cdot u_{xxy} + \frac{\Delta x \Delta y^2}{2} \cdot u_{xyy} - \\ - \frac{\Delta y^3}{6} \cdot u_{yyy} + \frac{\Delta x^4}{24} \cdot u_{xxxx} - \frac{\Delta x^3 \Delta y}{6} \cdot u_{xxxy} + \\ + \frac{\Delta x^2 \Delta y^2}{4} \cdot u_{xxyy} - \frac{\Delta x \Delta y^3}{6} \cdot u_{xyyy} + \frac{\Delta y^4}{24} \cdot u_{yyyy} + \ldots, \tag{K-3.10}$$

$$u_{i-1,j-1} = u_{ij} - \Delta x \cdot u_x - \Delta y \cdot u_y + \frac{\Delta x^2}{2} \cdot u_{xx} + \Delta x \Delta y \cdot u_{xy} + \\ + \frac{\Delta y^2}{2} \cdot u_{yy} - \frac{\Delta x^3}{6} \cdot u_{xxx} - \frac{\Delta x^2 \Delta y}{2} \cdot u_{xxy} - \frac{\Delta x \Delta y^2}{2} \cdot u_{xyy} - \\ - \frac{\Delta y^3}{6} \cdot u_{yyy} + \frac{\Delta x^4}{24} \cdot u_{xxxx} + \frac{\Delta x^3 \Delta y}{6} \cdot u_{xxxy} + \\ + \frac{\Delta x^2 \Delta y^2}{4} \cdot u_{xxyy} + \frac{\Delta x \Delta y^3}{6} \cdot u_{xyyy} + \frac{\Delta y^4}{24} \cdot u_{yyyy} + \ldots, \tag{K-3.11}$$

The modified equation for (K-3.7) is the following

$$u_{xx} + u_{xy} + u_{yy} = -\frac{1}{12}\left\{\left[u_{xxxx} + 8u_{xxxy}\right](\Delta x)^2 + \right. \\ \left. + \left[u_{yyyy} + 8u_{xyyy}\right](\Delta y)^2\right\} + \ldots . \tag{K-3.12}$$

From (K-3.12) follows that the approximation deficiency is $\approx O((\Delta x)^2, (\Delta y)^2)$.

5. Use five-point FDA for the second order derivative

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{ij} = \frac{-u_{i+2,j} + 16u_{i+1,j} - 30u_{i,j} + 16u_{i-1,j} - u_{i-2,j}}{12(\Delta x)^2}, \tag{K-3.13}$$

which, with substitution in the equation $u_{xx} + u_{yy} = x + y$, results in the following FDE:

$$\frac{-u_{i+2,j} + 16u_{i+1,j} - 30u_{i,j} + 16u_{i-1,j} - u_{i-2,j}}{12(\Delta x)^2} + \\ + \frac{-u_{i,j+2} + 16u_{i,j+1} - 30u_{i,j} + 16u_{i,j-1} - u_{i,j-2}}{12(\Delta y)^2} = i\Delta x + j\Delta y. \tag{K-3.14}$$

To develop the modified equation for FDE (K-3.14), use the Taylor series at the nodes $u_{i+2,j}; u_{i-2,j}; u_{i,j+2}; u_{i,j-2}$ with regards to the central node (i,j):

$$u_{i+2,j} = u_{ij} + 2\Delta x \cdot u_x + 2\Delta x^2 \cdot u_{xx} + \frac{4\Delta x^3}{3} \cdot u_{xxx} + \\ + \frac{2\Delta x^4}{3} \cdot u_{xxxx} + \frac{4\Delta x^5}{15} \cdot u_{xxxxx} + \frac{4\Delta x^6}{45} \cdot u_{xxxxxx} + \ldots \tag{K-3.15}$$

$$u_{i-2,j} = u_{ij} - 2\Delta x \cdot u_x + 2\Delta x^2 \cdot u_{xx} - \frac{4\Delta x^3}{3} \cdot u_{xxx} + \\ + \frac{2\Delta x^4}{3} \cdot u_{xxxx} - \frac{4\Delta x^5}{15} \cdot u_{xxxxx} + \frac{4\Delta x^6}{45} \cdot u_{xxxxxx} + \ldots \quad \text{(K-3.16)}$$

$$u_{i,j+2} = u_{ij} + 2\Delta y \cdot u_y + 2\Delta y^2 \cdot u_{yy} + \frac{4\Delta y^3}{3} \cdot u_{yyy} + \\ + \frac{2\Delta y^4}{3} \cdot u_{yyyy} + \frac{4\Delta y^5}{15} \cdot u_{yyyyy} + \frac{4\Delta y^6}{45} \cdot u_{yyyyyy} + \ldots \quad \text{(K-3.17)}$$

$$u_{i,j-2} = u_{ij} - 2\Delta y \cdot u_y + 2\Delta y^2 \cdot u_{yy} - \frac{4\Delta y^3}{3} \cdot u_{yyy} + \\ + \frac{2\Delta y^4}{3} \cdot u_{yyyy} - \frac{4\Delta y^5}{15} \cdot u_{yyyyy} + \frac{4\Delta y^6}{45} \cdot u_{yyyyyy} + \ldots \quad \text{(K-3.18)}$$

Use also the equations (K-3.2)-(K-3.5) and obtain the modified equation:

$$u_{xx} + u_{yy} = \frac{1}{90}\left(u_{xxxxxx}(\Delta x)^4 + u_{yyyyyy}(\Delta y)^4\right) + \ldots \quad \text{(K-3.19)}$$

The approximation deficiency in correspondence with (K-3.19) is $\approx O(\Delta x^4,\ \Delta y^4\ )$.

6. Biharmonic equation $\Delta^2 u = 0$, where Δ is Laplace's operator, can be rewritten in the form

$$\left(\partial^2/\partial x^2 + \partial^2/\partial y^2\right)^2 u = \left(\partial^2/\partial x^2 + \partial^2/\partial y^2\right)\left(\partial^2/\partial x^2 + \partial^2/\partial y^2\right)u = 0 \quad \text{or} \quad u_{xxxx} + 2u_{xxyy} + u_{yyyy} = 0. \quad \text{(K-3.20)}$$

Using the five-point approximation for the fourth order derivative:

$$\left(\frac{\partial^4 u}{\partial x^4}\right)_{ij} = \frac{u_{i+2,j} - 4u_{i+1,j} + 6u_{i,j} - 4u_{i-1,j} + u_{i-2,j}}{\Delta x^4}, \quad \text{(K-3.21)}$$

$$\left(\frac{\partial^4 u}{\partial y^4}\right)_{ij} = \frac{u_{i,j+2} - 4u_{i,j+1} + 6u_{i,j} - 4u_{i,j-1} + u_{i,j-2}}{\Delta y^4}, \quad \text{(K-3.22)}$$

compute

$$\left(\frac{\partial^4 u}{\partial x^2 \partial y^2}\right)_{ij} = \frac{\partial^2}{\partial x^2}\left(\frac{\partial^2 u}{\partial y^2}\right)_{ij} = \\ = \frac{u_{i-1,j-1} - 2u_{i-1,j} + u_{i-1,j+1} - 2u_{i,j-1} + 4u_{i,j} - 2u_{i,j+1}}{\Delta x^2 \Delta y^2} + \\ + \frac{u_{i+1,j-1} - 2u_{i+1,j} + u_{i+1,j+1}}{\Delta x^2 \Delta y^2}. \quad \text{(K-3.23)}$$

Now substituting the equations (K-3.21)-(K-3.23) into (K-3.20) yields:

$$\frac{u_{i+2,j}-4u_{i+1,j}+6u_{i,j}-4u_{i-1,j}+u_{i-2,j}}{\Delta x^4}+$$
$$\frac{u_{i-1,j-1}-2u_{i-1,j}+u_{i-1,j+1}-2u_{i,j-1}+4u_{i,j}-2u_{i,j+1}}{\Delta x^2\Delta y^2}+ \qquad \text{(K-3.24)}$$
$$+\frac{u_{i+1,j-1}-2u_{i+1,j}+u_{i+1,j+1}}{\Delta x^2\Delta y^2}+$$
$$+\frac{u_{i,j+2}-4u_{i,j+1}+6u_{i,j}-4u_{i,j-1}+u_{i,j-2}}{\Delta y^4}=0.$$

7. Supposed that the variables have been normalized in such a way that the box size is 1x1 and the slip velocity is *U=1*, introduce the coordinate system *OXY* as it is shown in Fig. K-3.2.

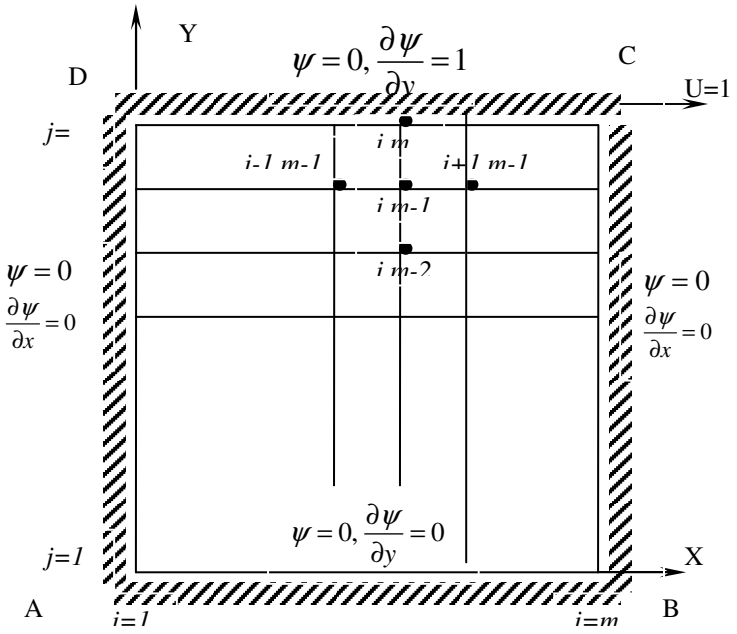


Fig. K-3.2. Boundary conditions and flow in quadratic domain.

If the liquid does not flow out of box due to the plate movement, the fluid flow appears to be just inside the box. Then the surfaces DA, AB, BC, CD are the segments of the boundary streamlines, where $\psi = 0$. By the streamline definition, there is $u=\partial\psi/\partial y,\ \ v=-\partial\psi/\partial x$ that means that the normal velocity component at these surfaces is zero. But for viscous liquid, the tangential velocity is also zero, except the moving plate CD where the velocity is 1. The basic and auxiliary boundary conditions are considered as it is shown in Fig. K-3.2.

Now biharmonic equation is reduced to the consequent Poisson's equation array

$$\Delta\zeta=0,\quad \Delta\psi=-\zeta\,, \qquad \text{(K-3.25)}$$

where is $\bar{\zeta} = (0,0,\zeta)$: $\bar{\zeta} = \nabla \times \bar{v}$, $\bar{v} = (u,v,w)$. Then generate the uniform quadratic numerical grid with the step *h=1/m*. The finite-difference approximation of the system (K-3.25) on the "criss-cross" mesh has been introduced earlier as (K-3.1). The boundary conditions for the second equation from the equation array (K-3.25) are clarified by Fig. K-3.2. Using all the boundary conditions one can state the boundary conditions for the first equation of the equation array (K-3.25). First an arbitrary point (*i,m*) on the top cover is considered (see Fig. K-3.2). Then applied the second equation of (K-3.25), one can suppose the following vortices in the point (*i,m*):

$$\zeta_{im} = -\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}\right) = \alpha_1\psi_{i-1,m-1} + \\ + \alpha_2\psi_{i,m-1} + \alpha_3\psi_{i+1,m-1} + \alpha_4\psi_{i,m-2} + \alpha_5\left(\frac{\partial\psi}{\partial y}\right)_{im} \qquad \text{(K-3.26)}$$

Now application of the Taylor series expansion results in

$$\psi_{i\pm1,m-1} = \psi_{im} \pm h\left(\frac{\partial\psi}{\partial x}\right)_{im} + \frac{h^2}{2}\left(\frac{\partial^2\psi}{\partial x^2}\right)_{im} - \\ - h\left(\frac{\partial\psi}{\partial y}\right)_{im} + \frac{h^2}{2}\left(\frac{\partial^2\psi}{\partial y^2}\right)_{im} + O\left(h^3\right), \qquad \text{(K-3.27)}$$

$$\psi_{i,m-k} = \psi_{im} - kh\left(\frac{\partial\psi}{\partial y}\right)_{im} + \frac{k^2h^2}{2}\left(\frac{\partial^2\psi}{\partial y^2}\right)_{im} + O\left(h^3\right), \qquad \text{(K-3.28)}$$

$k = 1, 2.$

Further substitution of (K-3.27), (K-3.28) into (K-3.26) yields with accuracy to $\approx O\left(h^3\right)$:

$$-\left(\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2}\right) = (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4)\psi_{im} + \\ + (\alpha_3 - \alpha_1)h\left(\frac{\partial\psi}{\partial x}\right)_{im} + \left(\frac{\alpha_5}{h} - \alpha_1 - \alpha_2 - \alpha_3 - 2\alpha_4\right)h\left(\frac{\partial\psi}{\partial y}\right)_{im} + \\ + (\alpha_1 + \alpha_3)\frac{h^2}{2}\left(\frac{\partial^2\psi}{\partial x^2}\right)_{im} + (\alpha_1 + \alpha_2 + \alpha_3 + 4\alpha_4)\frac{h^2}{2}\left(\frac{\partial^2\psi}{\partial y^2}\right)_{im}. \qquad \text{(K-3.29)}$$

The coefficients α_i *(i=1,2,3,4,5)* are calculated equated the same terms in (K-3.29) to the left and to the right. Then substitution of α_i in (K-3.26) yields:

$$\zeta_{im} = \frac{1}{h^2}\left(-\psi_{i-1,m-1} + \frac{8}{3}\psi_{i,m-1} - \psi_{i+1,m-1} - \frac{2}{3}\psi_{i,m-2}\right) + \frac{2}{3h}\left(\frac{\partial\psi}{\partial y}\right)_{im}. \qquad \text{(K-3.30)}$$

Note that in accordance with the auxiliary boundary conditions in (K-3.30) should be $\left(\frac{\partial\psi}{\partial y}\right)_{im} = 1.$ By analogy with the above-considered the following equations yield:

$$\zeta_{1,j} = \frac{1}{h^2}\left(-\psi_{2,j-1} + \frac{8}{3}\psi_{2,j} - \psi_{2,j+1} - \frac{2}{3}\psi_{3,j}\right), \tag{K-3.31}$$

$$\zeta_{m,j} = \frac{1}{h^2}\left(-\psi_{m-1,j-1} + \frac{8}{3}\psi_{m-1,j} - \psi_{m-1,j+1} - \frac{2}{3}\psi_{m-2,j}\right), \tag{K-3.32}$$

$$\zeta_{i,1} = \frac{1}{h^2}\left(-\psi_{i-1,2} + \frac{8}{3}\psi_{i,2} - \psi_{i+1,2} - \frac{2}{3}\psi_{i,3}\right). \tag{K-3.33}$$

Now the boundary conditions thus obtained for the flow vortices allow starting the solution of the boundary problem for the first equation (K-3.25) supposing that function ψ is known in some internal points of the domain. But the calculation of the function ψ from the second equation of the equation array (K-3.25) depends on ζ distribution inside the numerical domain. Thus, ψ and ζ are interconnected and the solution of the boundary problem must be found by iterations. First $\psi=0$ supposed to be in the whole flow domain. Then the boundary conditions for the vortices are determined from (K-3.30)-(K-3.33). And the boundary problem for ζ is solved indicating that the initial vortices supposed to be induced by top cover movement. The vortex calculated in such a way is used further for the elaboration of ψ by solution of the corresponding boundary problem. The next iteration repeats the consequent solution of the boundary problems for ζ and ψ. At each iteration in all points of the numerical grid, the difference between calculated values ζ and ψ and their values at the previous iteration is registered as local error. The sums of the absolute errors by both variables in all grid points are marked as ***erzeta*** and ***erpsi***, respectively. The iterations are going before the both *erzeta* and *erpsi* become less than stated value ***eps*** (required accuracy of the solution). The solution of the boundary problem for the Poisson's equation is performed by iterative Jacobi procedure (subroutine lieb). The listing of the computer code written in Fortran-90 is given below with all comments.

```
      Program steq

      implicit none

!  The program computes velocity field (streamline function)
!      and vorticity lines in quadratic domain

  integer,parameter  :: m = 11

! Dimension array
!              iter - number of iterations
      integer i,j,k,iter,mm1
  real,parameter  :: eps = 0.001
! Fidelity of calculation
!              h   - grid step, hsq = h*h
!      erzeta - the total error in the grid nodes by zeta
!      erpsi  - the total error in the grid nodes by psi
      real h,erzeta,erpsi,hsq
!      psi(m,m),psi1(m,m)  - arrays of data by psi
!      zeta(m,m),zeta1(m,m)- arrays of data by zita
!      rhs(m,m) - data arrays for right hand of Poisson equation
      real,dimension(m,m):: psi,psi1,zeta,zeta1,rhs
!      xx(m),yy(m) - arrays of data x,y
```

```
      real,dimension(m)  :: xx,yy
!     Open file of the results 'Psi.rez'
      open(10,file='Psi.rez')
!     Insert the initial data
      mm1=m-1
      iter=0
      h=1.0/mm1 ; hsq=h*h
      do i=1,m
            xx(i) = (i-1)*h
      end do
      do j=1,m
            yy(j) = (j-1)*h
      end do

!     Assumption about stationary flow
      psi = 0.0
      zeta= 0.0
!     Begin of the iteration procedure
1     iter=iter+1

!     Calculation of the boundary conditions by zeta
      do j=2,mm1
zeta(1,j)=(-psi(2,j+1)+8./3.*psi(2,j)-psi(2,j-1)     &
                     -2./3.*psi(3,j))/hsq
zeta(m,j)=(-psi(mm1,j+1)+8./3.*psi(mm1,j)            &
            -psi(mm1,j-1)-2./3.*psi(m-2,j))/hsq
      end do
      do i=2,mm1
zeta(i,1)=(-psi(i+1,2)+8./3.*psi(i,2)-psi(i-1,2)     &
                      -2./3.*psi(i,3))/hsq
zeta(i,m)=(-psi(i+1,mm1)+8./3.*psi(i,mm1) &
      -psi(i-1,mm1)-2./3.*psi(i,m-2))/hsq- 2./(3.*h)
      end do
!     Reinsert data before iteration by zeta
      rhs=0.0
      zeta1=zeta
!     Call the Jacobi procedure
      call lieb(zeta,rhs,m,eps)

!     Absolute error calculation
      erzeta=0.0
      do i=2,mm1
            do j=2,mm1
      erzeta=erzeta+abs(zeta(i,j)-zeta1(i,j))
            end do
      end do

! Reinsert data before iteration by psi
            rhs =-zeta
            psi1= psi
!     Call the Jacobi procedure
      call lieb(psi,rhs,m,eps)
!     Calculation of absolute error
      erpsi=0.0
      do i=2,mm1
            do j=2,mm1
      erpsi=erpsi+abs(psi(i,j)-psi1(i,j))
            end do
      end do
!     Output of intermediate information on the screen
      write(*,10) iter,erpsi,erzeta

!     Output of the information in external file 'Psi.rez'
      write(10,10) iter,erpsi,erzeta
```

```
10      format(//10x,'iter =',i5,' erpsi =',f10.5,' & erzeta =',f10.3)

!       Comparison of the absolute error with given limit
        if((erzeta.gt.eps).or.(erpsi.gt.eps)) goto 1
!       The end of iterations

!       Calculation of the vorticity in the corners of the domain
        zeta(1,1)=(zeta(1,2)+zeta(2,1))/2.0
        zeta(m,1)=(zeta(m,2)+zeta(mm1,1))/2.0
        zeta(1,m)=(zeta(2,m)+zeta(1,mm1))/2.0
        zeta(m,m)=(zeta(m,mm1)+zeta(mm1,m))/2.0

!       Output of the information in external file 'Psi.rez'
        write(10,'(7x,11f6.3)') (yy(i),i=1,m)
        do k=1,m
                j=m-k+1
                write(10,11) xx(j),(psi(i,j),i=1,m)
        end do
11      format(1x,12f6.3)
        close(10)
!
!       Output of the information in external file 'Zita.rez'
        open(11,file='Zita.rez')
        write(11,12) erzeta
!
12      format(///28x,'Vorticity distribution'/ &
                32x,'erzeta =',f7.5)
        write(11,'(7x,11f6.3)') (yy(i),i=1,m)
        do k=1,m
                j=m-k+1
                write(11,11) xx(j),(zeta(i,j),i=1,m)
        end do
        close(11)
!
        contains
!
!       The iterative Jacobi procedure
        Subroutine lieb(f,q,mm,err)
        integer mm
        real fold,error,err
        real,dimension(mm,mm) :: f,q

!       Statement of initial data
        do i=2,mm1
               do j=2,mm1
                f(i,j)=0.0
        end do
        end do

!       Start the internal iterative procedure
2       error=0.0
        do i=2,mm1
                do j=2,mm1
                fold=f(i,j)
           f(i,j)=0.25*(f(i-1,j)+f(i+1,j)+f(i,j-1)+f(i,j+1) &
                -h*h*q(i,j))
         error=error+abs(f(i,j)-fold)
                end do
        end do
        if(error.gt.err) goto 2
        return
        end Subroutine lieb
        End Program steq
```

8. By introduction of the moving coordinate system shown in Fig.K-3.3 (the axis *OX* coincides with a symmetry axis of the channel), the task is simply reduced to the previous one. One just needs to apply appropriate non-dimensionalization.

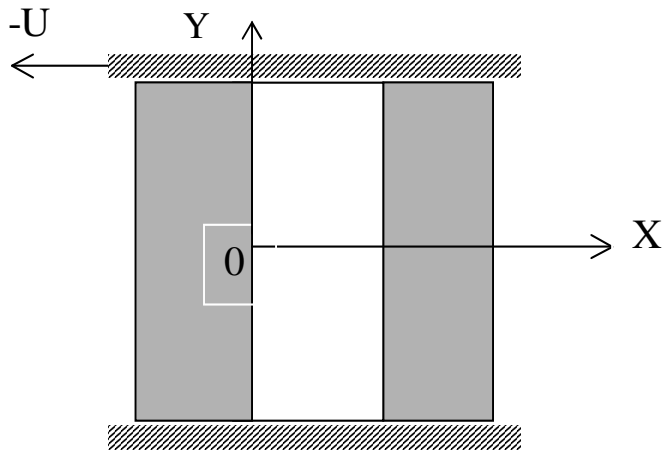


Fig. K-3.3. Boundary conditions and flow domain.

Note: though boundary conditions for ψ on the axis *OX* look as boundary condition on AB in previous task (Fig. K-3.3.), their physical content is quite different. Here it is due to the system symmetry with regards to the gap axis while in the previous task zero velocity was stated at the rigid body due to viscosity. Gauss-Seidel method with Sequential Upper Relaxation is used here. Therefore the main procedure in computer code is replaced (subroutine lieb). All the comments are written in the program listing, which is given below.

```
    Program Cells

    implicit none
!   The program calculates velocity field (stream function) and
!   vorticity between blood cells by the Gauss-Seidel method with
!   Sequential upper relaxation

    integer,parameter :: m = 11
!   Array dimension
!          iter - number of iterations
    integer i,j,k,iter,mm1
    real,parameter  :: eps = 0.001
!   Fidelity of calculations
    real,parameter  :: rf  = 1.00
!   Relaxation parameter
!          h   - grid step, hsq = h*h
!    erzeta - the total error in grid nodes by zeta
!    erpsi  - the total error in grid nodes by psi
    real h,erzeta,erpsi,hsq
!   psi(m,m),psi1(m,m)  - data arrays of psi
!   zeta(m,m),zeta1(m,m)- data arrays of zita
!   rhs(m,m) - array of the right hand of the Poisson equation
```

```
      real,dimension(m,m):: psi,psi1,zeta,zeta1,rhs
!     xx(m),yy(m) - data arrays of x,y
      real,dimension(m)  :: xx,yy
      open(10,file='Psi-Cell.rez')
!     Insert initial data
      mm1=m-1
      iter=0
      h=1.0/mm1 ; hsq=h*h
      do i=1,m
              xx(i) = (i-1)*h
      end do
      do j=1,m
              yy(j) = (j-1)*h
      end do
!     Assumption about stationary flow
      psi = 0.0
      zeta= 0.0
!
!     Begin of iterative procedure
1     iter=iter+1
!
!     Calculation of boundary conditions for zeta
      do j=2,mm1
zeta(1,j)=(-psi(2,j+1)+8./3.*psi(2,j)-psi(2,j-1)      &
                    -2./3.*psi(3,j))/hsq
zeta(m,j)=(-psi(mm1,j+1)+8./3.*psi(mm1,j)-psi(mm1,j-1)&
                    -2./3.*psi(m-2,j))/hsq
      end do
      do i=2,mm1
zeta(i,1)=(-psi(i+1,2)+8./3.*psi(i,2)-psi(i-1,2)      &
                     -2./3.*psi(i,3))/hsq
zeta(i,m)=(-psi(i+1,mm1)+8./3.*psi(i,mm1)-psi(i-1,mm1)  &
                     -2./3.*psi(i,m-2))/hsq- 2./(3.*h)
      end do
!
!     Reinsert data before new iteration by zeta
      rhs=0.0
      zeta1=zeta

!     Call Gauss-Seidel procedure with Sequential Upper Relaxation
      call lieb(zeta,zeta1,rhs,m,eps)

!     Calculation of absolute error
      erzeta=0.0
      do i=2,mm1
              do j=2,mm1
      erzeta=erzeta+abs(zeta(i,j)-zeta1(i,j))
              end do
      end do

!     Reinsert data before new iteration by psi
              rhs =-zeta
              psi1= psi

!     Call the procedure Gauss-Seidel with Sequential Upper Relaxation
      call lieb(psi,psi1,rhs,m,eps)

!     Calculation of absolute error
      erpsi=0.0
      do i=2,mm1
              do j=2,mm1
      erpsi=erpsi+abs(psi(i,j)-psi1(i,j))
              end do
      end do
```

```
!   Output of intermediate data on the screen
    write(*,10) iter,erpsi,erzeta
!   Output of the data in external file 'Psi-Cell.rez'
    write(10,10) iter,erpsi,erzeta
10  format(//10x,'iter =',i5,' erpsi =',f10.5,' erzeta =',f10.3)

!   Comparison of absolute error with the limit stated
    if((erzeta.gt.eps).or.(erpsi.gt.eps)) goto 1
!   The end of iterative procedure
!
!   Calculation of the vorticity in the corners of domain
    zeta(1,1)=(zeta(1,2)+zeta(2,1))/2.0
    zeta(m,1)=(zeta(m,2)+zeta(mm1,1))/2.0
    zeta(1,m)=(zeta(2,m)+zeta(1,mm1))/2.0
    zeta(m,m)=(zeta(m,mm1)+zeta(mm1,m))/2.0

!   Output of the data in external file 'Psi-Cell.rez'
    write(10,'(7x,11f6.3)') (yy(i),i=1,m)
    do k=1,m
            j=m-k+1
            write(10,11) xx(j),(psi(i,j),i=1,m)
    end do
11  format(1x,12f6.3)
    close(10)

!   Output of the data in external file 'Zita-Cell.rez'
    open(11,file='Zita-Cell.rez')
    write(11,12) erzeta
12  format(///28x,'Vorticity distribution'/ &
            32x,'erzeta =',f7.5)
    write(11,'(7x,11f6.3)') (yy(i),i=1,m)
    do k=1,m
            j=m-k+1
            write(11,11) xx(j),(zeta(i,j),i=1,m)
    end do
    close(11)
!
    contains
!
!   Subroutine for calculation of vorticity and stream function
!   by Gauss-Seidel procedure with Sequential Upper Relaxation
    subroutine lieb(f,f1,q,mm,err)
    integer mm
    real fold, fold1,err,error
    real,dimension(mm,mm):: f,f1,q

!   Insert data of function f in the internal grid nodes
    do i=2,mm1
            do j=2,mm1
            f(i,j)=0.0
            end do
    end do
2   error=0.0
    do i=2,mm1
            do j=2,mm1
            fold=f(i,j)
  fold1=0.25*(f1(i-1,j)+f(i+1,j)+f1(i,j-1)+f(i,j+1) &
       -h*h*q(i,j))

!   Gauss-Seidel procedure by rf=1.0 and Sequential Upper Relaxation
!   procedure by 0<rf<2
    f(i,j)=rf*fold1+(1.-rf)*f1(i,j)
    error=error+abs(f(i,j)-fold)
```

```
        end do
end do
if(error.gt.err) goto 2
return
end subroutine lieb
End Program Cells
```

Choose the relaxation parameter and prove: the Sequential Upper Relaxation procedure speeds the convergence up.

9. The Newton's method (see the scheme in Fig. 3.2) applied to the equation array (E-3.1) at each iterative step k requires the solution of the equation with regards to $\Delta\vec{T}^{(k+1)}$:

$$\underline{J}^{(k)}\Delta\vec{T}^{(k+1)} = -\vec{R}^{(k)}\ , \tag{K-3.34}$$

where $\vec{R}^{(k)} = \left\{R_1^{(k)}, R_2^{(k)}, R_3^{(k)}\right\}$, $\vec{T}^{(k)} = \left\{T_1^{(k)}, T_2^{(k)}, T_3^{(k)}\right\}$. The matrix components $\underline{J}^{(k)} = \left\{J_{ij}^{(k)} = \partial R_i^{(k)} / \partial T_j^{(k)}\right\}$ are:

$$\underline{J} = \begin{bmatrix} \left(4T_1^3 + C_1\right) & -\left(4T_2^3 + C_0\right) & 0 \\ \left(4T_1^3 + C_0\right) & -\left(8T_2^3 + C_2\right) & \left(4T_3^3 + C_0\right) \\ 0 & \left(4T_2^3 + C_0\right) & -\left(8.2 \cdot T_3^3 + C_3\right) \end{bmatrix} \tag{K-3.35}$$

Here are: C_0=0.05848; C_1=0.06823; C_2=0.11696; C_3=0.2534. The solution of the system (E-3.1) on $(k+1)$-th iteration is calculated as follows

$$\vec{T}^{(k+1)} = \vec{T}^{(k)} + \Delta\vec{T}^{(k+1)}. \tag{K-3.36}$$

The solution of the algebraic equation array by the Gauss method is performed using the procedure LSLRG from standard library IMSL (MSIMSL for MS FPS 4.0 or DFIMSL for CVF 6.0-6.5). The listing of computer code written in FORTRAN-90 is given below.

```
      Program Newton

!     Newton's method for non-linear algebraic equation array R(T)=0
!     use dfimsl
!     use IMSL library
!
      Implicit none
!     n - number of equations
      integer, parameter :: n = 3
!     itmx - max number of iterations
      integer, parameter :: itmx = 100
!     iter - number of iterations
      integer iter,i,j
!     eps  - max computational error
      real, parameter :: eps = 0.00001
!     rmsr - computational error
      real*8 rmsr,sum,aN
!     T(n),T0(n) - arrays of temperature
!     R(n) - array of residual
      real,dimension(n):: T,T0,R,dT
!     AJ(n,n) - jacobian
      real,dimension(n,n):: AJ
      external RESID,JACOB
!
```

```
!   Zero approximation
    T0 = 0.3
!   open the file "Newton.rez"
    open(2,file='Newton.rez')
!   testing data
    write(2,12) n,itmx,eps
12  format(/,' Newton"s method for n=',i3,  &
    ' itmx= ',i3,' eps= ',e10.3)
    write(2,13) (T0(j),j=1,n)
13  format(' Zero approximation To = ',3f5.2)
!
    aN = n
    iter = 0
!
!   Start iteration
    do
    iter = iter + 1

!   Calculate residuals at point To
    call RESID(n,T0,R)

!   Calculate computational error
    sum= 0.0d0
    do i=1,n
        sum = sum + R(i)*R(i)
    end do
    rmsr = dsqrt(sum/aN)

!   CRT screen output
    write(*,14) rmsr
!   Output into file
    write(2,14) rmsr
14  format(/,' computational error rmsr = ',d11.4)
!   Thresholding validity
    if((rmsr .LT. eps) .OR. (iter .GE. itmx))  exit
!
!   Calculate jacobian
    call JACOB(n,T0,AJ)

!   Calculate SLAE
    call Lslrg(n,AJ,n,-R,1,dT)
!   Increment T
    T = T0 + dT
!   Re-change T
    T0 = T
!   End iteration
    end do
!
!   Generate output
    write(2,15) iter,rmsr
15  format(/,' after ',i3,' iteration rmsr = ',d11.4)
    write(2,16) (T(j),j=1,n)
16  format(/,' T = ',1pE16.9)
    stop
    End  Program Newton

    Subroutine RESID(nn,T,R)
!   Evaluates residuals required by Newton's method
!
    Implicit none
!
    Integer j,nn
    Real c,dum,dam,T4(10)
    Real,dimension(nn):: T,R
```

```
!
    T4 = 0.0
    do j=1,nn
          dum = T(j)
          T4(j) = dum*dum*dum*dum
    end do

    c = 0.05848
    dum = T4(2) + c*T(2)
    dam = T4(3) + c*T(3)
    R(1) = T4(1) + 0.06823*T(1) - dum - 0.01509
    R(2) = T4(1) + c*T(1) - 2.0*dum + dam
    R(3) = dum - 2.05*T4(3) - 0.2534*T(3) + 0.06698
    Return
    End Subroutine RESID

    Subroutine JACOB(nn,T,AJ)
!   Evaluates Jacobian required by Newton's method
!
    Implicit none
   Integer j,nn
    Real c,dum,T3(10)
    Real,dimension(nn):: T
    Real,dimension(nn,nn):: AJ
!
    AJ = 0.0
    c = 0.05848
    do j = 1,nn
        dum = T(j)
        T3(j) = 4.*dum*dum*dum
    end do

    AJ(1,1) = T3(1) + 0.06823
    AJ(1,2) = -T3(2) - c
    AJ(2,1) = T3(1) + c
    AJ(2,2) = -2.0*T3(2) - 0.11696
    AJ(2,3) = T3(3) + c
    AJ(3,2) = T3(2) + c
    AJ(3,3) = -2.05*T3(3) - 0.2534
    Return
    End Subroutine JACOB
```

10. The solution of the equation (K-3.34) with regards to $\Delta\vec{T}^{(k+1)}$ yields:

$$\Delta\vec{T}^{(k+1)} = -\left[\underline{J}^{(k)}\right]^{-1}\vec{R}^{(k)}.$$

Then the replacement $\left[\underline{J}^{(k)}\right]^{-1} \to \underline{H}^{(k)} : H_{ii}^{(k)} = 1/J_{ii}^{(k)}, \quad H_{ij}^{(k)} = 0, \quad i \neq j$ results in

$$\Delta\vec{T}^{(k+1)} = -\underline{H}^{(k)}\vec{R}^{(k)}. \qquad \text{(K-3.37)}$$

Hence the matrix $\underline{H}^{(k)}$ is diagonal, the solution of (K-3.37) is

$$\Delta\vec{T}_i^{(k+1)} = -R_i^{(k)}/J_{ii}^{(k)}. \qquad \text{(K-3.38)}$$

Make the changes in the computer program from the task 9 and compare the solution obtained with the solution of the task 9 (by calculation time required to achieve the same fidelity in both solutions).

Chapter 4

1. The stability condition for the LAEA with tridiagonal matrix

$$a_n T_{j-1}^{n+1} + b_n T_j^{n+1} + c_n T_{j+1}^{n+1} = d_n \qquad \text{(K-4.1)}$$

is following:

$$|b_n| \geq |a_n| + |c_n|, \quad n = \overline{2, N-1}. \qquad \text{(K-4.2)}$$

Such number n_1 must exist so that $|b_{n_1}| > |a_{n_1}| + |c_{n_1}|$ is satisfied. Thus, the condition (K-4.2) has to be proved for all known implicit schemes:

a) Fully implicit difference scheme (FI):

$$b_n = 1 + 2s,\ a_n = c_n = -s,\ s = \alpha \Delta t / \Delta x^2 .$$

Then $|1 + 2s| > |s| + |s|$ and stability condition satisfies by any $s>0$.

b) Crank-Nicholson scheme (CN):

$$b_n = 1 + s, \quad a_n = c_n = -s/2 .$$

Then $|1 + s| > 1/2|s| + 1/2|s|$ and stability condition satisfies by any $s>0$.

c) Generalized three-layer implicit scheme (G3L):

$$b_n = 1 + \gamma + s \cdot 2\beta, \quad a_n = c_n = -s \cdot \beta .$$

The stability condition $|1 + \gamma + s \cdot 2\beta| > |s \cdot \beta| + |s \cdot \beta|$ is satisfied for $\beta \geq 1/2$ and $\gamma, s > 0$.

2. Algorithm (4.8) is used for calculation. At each time-step, the solution is computed until the numbers Nmax or TiMax are achieved. The numerical solution is compared to the exact analytical solution

$$T_E(t,x) = 100 - \sum_{m=1}^{MaxEx} \left[\frac{400}{(2m-1)\pi} \right] \sin\left[(2m-1)\pi x \right] e^{-\alpha (2m-1)^2 \pi^2 t}, \qquad \text{(K-4.3)}$$

obtained by the method of separated variables. The mean-square error is

$$rms = \left[\left(\sum_{j=1}^{J\max} \left(T_j - TE_j \right)^2 \right) \Big/ J\max \right]^{1/2}. \qquad \text{(K-4.4)}$$

The listing of the FORTRAN-90 computer program with comments is given below.

```
      Program Diff
!     Solves 1D transient heat conduction using FTCS scheme
```

```
!     Jmax - max point number along the rod
      Integer,Parameter :: Jmax = 11
!     Nmax - max number of time steps
      Integer,Parameter :: Nmax = 500
!    MaxEx - max number of terms in the exact solution
      Integer,Parameter :: MaxEx = 10
      Integer Jmap,n,j,m
!
      Real,Parameter :: Pi = 3.1415927
      Real,Parameter :: dT = 500.0    ! Time step
      Real,Parameter :: dX = 0.1      ! Step along the rod
      Real,Parameter :: tMax = 5999.0 ! Max Time
!     t - Time
!     Alph - the thermal diffusivity
      Real Alph,t,s,ajm,sum,dam,am,avs,rms,s0,dxm
!     X(*)  - coordinate along the rod
!     TN(*) - nondimensional temperature
!     TD(*) - dimensional temperature
!    TE(*)- dimensional temperature (exact solution)
      Real,Dimension(Jmax) :: TN,TD,TE,X

!     Input data
         s = 0.5
!        s = 1./6.
      If(s .GT. 0.5) Then
     Write(*,'(/a,F8.5,a\)') ' s= ',s,'Input less s ='
        Read(*,*) s0
        s = s0
                    End If
      Jmap = Jmax - 1
       ajm = Jmap
      Alph = s*dX*dX/dT
      Open(2,file = 'Diff.rez')
!     Write input data into file = 'Diff.rez'
      Write(*,100) Jmax,MaxEx,Nmax,TMax
      Write(2,100) Jmax,MaxEx,Nmax,TMax
100   Format(/' Jmax = ',i3,' MaxEx = ',i3, &
        ' Nmax = ',i3,' tMax = ',1pE12.5)
      Write(*,101) s,Alph,dT,dX
      Write(2,101) s,Alph,dT,dX
101   Format(/' s= ',1pE11.4,' Alpha= ',1pE11.4, &
      ' dT = ',1pE11.4,' dX = ',1pE11.4,/)
!
!     Set initial condition
      do j = 1,Jmap
        TN(j) = 0.0
      end do
      t = 0.0
      n = 0
!     Start iterations
      do
!     Set boundary conditions
!
      TN(1) = 1.0
      TN(Jmax) = 1.0
      If(t .LT. 0.01) Then
        TN(1) = 0.5
          TN(Jmax) = 0.5
                    End If
!
!     Compute FTCS solution
      Call FTCS
!
!     Obtain dimensional temperature
```

```
      TD = 100.0*TN
      t = t + dT
      n = n + 1
!     Write rezult into file = 'Diff.rez'
      Write(2,102) t,(TD(j),j=1,Jmax)
102   Format(' T= ',F5.0,' TD= ',11F7.2)
!
!     If maximal time or maximal number of time-steps is
!     exceeded, then exit the block
      if((t.GE.TMax) .OR. (n.GE.Nmax)) exit
      end do

!     Obtain the exact solution and compare
      sum = 0.0
      do j = 1,Jmax
           aj = j - 1
         X(j) = dX*aj
        TE(j) = 100.0
        do m = 1,MaxEx
                 am = m
                dam = 2.*am-1
                dxm = dam*Pi*X(j)
                dtm = -Alph*dam*dam*Pi*Pi*t
!
!     Limit the argument size of exp(dtm)
      If(dtm .LT. -87.0) dtm = -87.0
    TE(j) = TE(j) - 400./dam/Pi*sin(dxm)*exp(dtm)
        end do
        sum = sum + (TE(j)-TD(j))**2
      end do
      Write(2,103) t,(TE(j),j=1,Jmax)
103   Format(/,' T= ',F5.0,' TE= ',11F7.2,//)
!
!     rms is the RMS error
      avs = sum/(1.+ ajm)
      rms = sqrt(avs)
      Write(2,104) rms
104   Format(' RMS DIF = ',1pE11.4,/)
      Close(2)
      Stop

      Contains

      Subroutine FTCS
!     FTCS procedure
      Integer i
      Real,dimension(41) :: DUM

      DUM = 0.0
      do i = 2,Jmap
        DUM(i)= (1.-2.*s)*TN(i)+s*(TN(i-1)+TN(i+1))
      end do
      do i = 2,Jmap
        TN(i) = DUM(i)
      end do
      End Subroutine FTCS
      End Program Diff
```

3. The mean-square errors *rms* for the temperature profile calculated by computer code `Diff` are given in the Table for the different steps Δx and parameters `s`:

	Mean-square errors *rms*			
Δx	$s = 0.5$	$s = 0.3$	s = 0.1	s = 1/6
0.1	3.4979E-01	3.0380E-01	2.7671E-01	9.1508E-03
0.2	7.0854E-01	2.0142E-01	4.3359E-01	4.3189E-03

Obviously the computation accuracy grows drastically (two orders at all) for `s=1/6`. Explanation of the phenomenon is supported by the analysis of modified equation for the FTCS scheme (see (4.9), p. 106). The underlined truncation error term is of order $\approx O\left(\Delta t,(\Delta x)^2\right)$ that remains in general by decrease of parameter `s`. Meanwhile by `s=1/6` the underlined term becomes zero. The whole accuracy of the numerical scheme grows in this case due to the new truncation error term that has accuracy $\approx O\left((\Delta t)^2,(\Delta x)^4\right)$.

4. The calculations are performed by the algorithms (3.8) and (3.12). The modified computer code Diff (in FORTRAN-90) with the user-friendly comments is given below.

```
Program Diff
!       Solves 1D transient heat conduction equation
!       using FTCS and DuFort-Frankel numerical schemes
!   Jmax - max number point along the rod
Integer,Parameter :: Jmax = 11
!       Nmax - max number of time steps
Integer,Parameter :: Nmax = 500
!       MaxEx- max number of terms in the exact solution
Integer,Parameter :: MaxEx = 10
!
Integer Jmap,n,j,Itip
!
Real,Parameter :: Pi = 3.1415927
!       Alph - the thermal diffusivity
Real,Parameter :: Alph = 0.01
Real,Parameter :: tMin = 2.0   ! Min Time
Real,Parameter :: tMax = 9.0   ! Max Time
!       t        - dimensionless time
!       dT     - time-step
!       tm       - dimensional time
!       dX       - step along the rod

Real dT,t,s,dX,ajm,as,bs
Real*8 sum,avs,rms,dmp
!       X(*)  - coordinate along the rod
!       TD(*) - dimensional temperature
!       TE(*) - dimensional temperature (exact solution)

Real,Dimension(Jmax) :: X,TD,TE,TN,TN0
!       TN(*,*) - nondimensional temperature
!       Real,Dimension(0:1,Jmax) :: TN
!       Input data
   Write(*,'(/a\)') ' Input Tip (1-FTCS,2-DF) ='
Read(*,*) Itip
       Jmap = Jmax - 1
     ajm = Jmap
```

```
          dX = 1.0/ajm
           s = 0.3
!           s = 1./sqrt(12.0)
          dT = dX*dX*s/Alph
as = 2.*s/(1.+2.*s)
bs = (1.-2.*s)/(1.+2.*s)
!
Open(2,file = 'Duf_Fr.rez')
!       Write input data into file = 'Duf_Fr.rez'
Write(*,100) Jmax,MaxEx,Nmax,TMax
Write(2,100) Jmax,MaxEx,Nmax,TMax
100     Format(/' Jmax = ',i3,' MaxEx = ',i3, &
!             ' Nmax = ',i3,' tMax = ',1pE12.5)
Write(*,101) s,Alph,dT,dX
Write(2,101) s,Alph,dT,dX
101     Format(/' s =',1pE11.4,' Alpha =',1pE11.4 &
       ,' dT = ',1pE11.4,' dX = ',1pE11.4,/)
!
!       Obtain initial condition from the exact solution
!
t = tMin - 2.*dT
Do n = 1,2
  t = t + dT
!
  Call EXACT
!
  do j = 2,Jmap
        If(n .EQ. 1) TN0(j) = TE(j)/100.0
        If(n .EQ. 2) TN(j)  = TE(j)/100.0
  end do
End Do
!       Set boundary conditions
   TN(1) = 1.0 ;    TN0(1) = 1.0
TN(Jmax) = 1.0 ; TN0(Jmax) = 1.0
If(t .LT. 0.01) Then
!       If(t .LT. 0.01 .AND. n .GT. 0) Then
       TN(1) = 0.5
    TN(Jmax) = 0.5
                    End If
n = 0
t = tMin
!
!       Start iterations
do
n = n + 1

!       Compute FTCS solution
!
If(Itip .EQ. 1) Call FTCS
!
!       Compute DuFort-Frankel solution
!
If(Itip .EQ. 2) Call Duf_Frank
!
!       Obtain dimensional temperature
TD = 100.0*TN
!
 t = t + dT
!       Write rezult into file = 'Duf_Fr.rez'
Write(2,102) t,(TD(j),j=1,Jmax)
102     Format(' T= ',F5.2,' TD= ',11F7.2)
!
!If maximal time or number of time-steps exceeded, exit the block
if((t.GE.TMax) .OR. (n.GE.Nmax)) exit
```

```
end do
!
!       Obtain exact solution and compare
sum = 0.0d0
!
Call EXACT
!
do j = 1,Jmax
        dmp = TE(j) - TD(j)
        sum = sum + dmp*dmp
end do
Write(2,103) t,(TE(j),j=1,Jmax)
103     Format(/,' T= ',F5.2,' TE= ',11F7.2,//)

!       rms is the RMS error
!
avs = sum/(1.+ ajm)
rms = dsqrt(avs)
Write(2,104) rms
104     Format(' RMS DIF = ',1pD11.4,/)
Close(2)
Stop

Contains

Subroutine FTCS
!       The FTCS procedure
Integer i
Real,dimension(Jmax) :: DUM
     DUM = 0.0
do i = 2,Jmap
    DUM(i) = (1.-2.*s)*TN(i)+s*(TN(i-1)+TN(i+1))
end do
do i = 2,Jmap
   TN(i) = DUM(i)
end do

End Subroutine FTCS

Subroutine Duf_Frank
!       The DuFort-Frankel procedure
Integer i
Real,dimension(Jmax) :: DUM
     DUM = 0.0
do i = 2,Jmap
  DUM(i) = as*(TN(i-1)+TN(i+1))+bs*TN0(i)
end do
     TN0 = TN
do i = 2,Jmap
   TN(i) = DUM(i)
end do

End Subroutine Duf_Frank

Subroutine EXACT
!  Exact solution of the transient heat conduction problem
!
Integer m,i
Real ai,dam,am,dxm,dtm
Real,Dimension(Jmax) :: X0
Do i = 1,Jmax
          ai = i - 1
         X0(i) = dX*ai
         TE(i) = 100.0
```

```
        do m = 1,MaxEx
                 am = m
                dam = 2.*am-1
                dxm = dam*Pi*X0(i)
                dtm = -Alph*dam*dam*Pi*Pi*t
!
!       Limit the argument size of exp(dtm)
        If(dtm .LT. - 25.0) dtm = - 25.0
                dtm = exp(dtm)
        If(dtm .GE. 1.0e-10) Then
TE(i) = TE(i) - 400./dam/Pi*sin(dxm)*dtm
        End If
        end do
End Do
End Subroutine EXACT
End Program Diff
```

The results of calculation by FTCS and DuFort-Frankel schemes are given in the Table:

Scheme	s	Mean-square error *rms*		
		Δx = 0.2	*Δx = 0.1*	*Δx = 0.05*
FTCS	1/6	7.2655D-03	4.4849D-04	1.1919D-05
	0.3	6.4388D-01	1.6340D-01	4.1315D-02
	0.41	1.2438D+00	3.0226D-01	7.5514D-02
DuFort-Frankel	1/√12	5.1408D-02	2.7934D-03	2.2928D-04
	0.3	2.4420D-02	1.3603D-02	3.9655D-03
	0.41	8.5260D-01	2.0849D-01	5.2502D-02

The peculiarities of the FTCS scheme have been discussed in detail in the previous task. The DuFort-Frankel scheme has higher accuracy than FTCS scheme but its advantages reveal more for the less time-steps. Nevertheless the positive circumstance is that the DuFort-Frankel scheme is more precise than FTCS even when Δx and Δt are of the same order (sometimes even by $\Delta t \geq \Delta x$). Note: the DuFort-Frankel scheme is convergent also for s>1/2. But it was shown in the previous task that FTCS is preferable for $\Delta t >> \Delta x$.

5. The solution algorithm is similar to the ones applied for the tasks 2-4. The difference consists in the statement of the initial and boundary conditions. Listing of the modified Fortran-90 program Diff is placed below with the user-friendly comments.

```
   Program Diff
!  Solves 1D transient heat conduction using
! FTCS, Dufort-Frankel or Barakat-Clark schemes
!   Jmax - max point number along the rod
   Integer,Parameter :: Jmax = 11
!  Nmax - max number of time steps
   Integer,Parameter :: Nmax = 500
! MaxEx-max number of terms in the exact solution
   Integer,Parameter :: MaxEx = 10
!
   Integer Jmap,n,j,Itip
```

```
!
   Real,Parameter :: Pi = 3.1415927
!  Alph - the thermal diffusivity
   Real,Parameter :: Alph = 0.02
   Real,Parameter :: Co   = 100.0 ! grad C
   Real,Parameter :: aL = 1.0 ! wall thickness
   Real,Parameter :: tMin = 0.0   ! Min Time
   Real,Parameter :: tMax = 10.0  ! Max Time
!  t        - nondimensional time
!  dT   - time step
!  tm       - dimensional time
!  dX       - step along the rod
   Real as1,as2,dT,t,s,dX,ajm,as,bs
   Real*8 sum,avs,rms,dmp
!  X(*)  - coordinate along the wall
!  TD(*) - dimensional temperature
! TE(*)- dimensional temperature (exact solution)
   Real,Dimension(Jmax) :: X,TD,TE,TN,TN0
!  TN(*,*) - non-dimensional temperature
!  Real,Dimension(0:1,Jmax) :: TN
!  Input data
Write(*,'(/a\)')' Input Tip(1-FTCS,2-DF,3-BC) = '
   Read(*,*) Itip
          Jmap = Jmax - 1
        ajm = Jmap
            dX = 1.0/ajm
            dT = 0.1
             s = dT*Alph/dX/dX
!           s = 1./sqrt(12.0)
!           dT = dX*dX*s/Alph
    as1 = (1.- s)/(1.+ s)
    as2 = s/(1.+ s)
    as = 2.*s/(1.+ 2.*s)
    bs = (1.- 2.*s)/(1.+ 2.*s)
!
   Open(2,file = 'Bar_Cl.rez')
!  Write input data into file = 'Bar_Cl.rez'
   Write(*,100) Jmax,MaxEx,Nmax,TMax
   Write(2,100) Jmax,MaxEx,Nmax,TMax
Format(/' Jmax = ',i3,' MaxEx = ',i3 &
,' Nmax = ',i3,' tMax = ',1pE12.5)
   Write(*,101) s,Alph,dT,dX
   Write(2,101) s,Alph,dT,dX
Format(/' s= ',1pE11.4,' Alpha= ',1pE11.4 &
,' dT = ',1pE11.4,' dX = ',1pE11.4,/)
!
!  Obtain the initial condition from the exact solution
!
   t = tMin - dT
   Do n = 1,2
     t = t + dT
!
     Call EXACT
     do j = 2,Jmap
          If(n .EQ. 1) TN0(j) = TE(j)/100.0
          If(n .EQ. 2) TN(j)  = TE(j)/100.0
     end do
   End Do
!
!  Set the boundary conditions
      TN(1) = 0.0 ;    TN0(1) = 0.0
   TN(Jmax) = 0.0 ; TN0(Jmax) = 0.0
   n = 0
   t = tMin
```

```
   If(Itip .EQ. 2) t = tMin + dT
!
!  Start iterations
   do
   n = n + 1
!  Compute FTCS solution
   If(Itip .EQ. 1) Call FTCS
!
!  Compute DuFort-Frankel solution
   If(Itip .EQ. 2) Call Duf_Frank
!
!  Compute Barakat-Clark solution
   If(Itip .EQ. 3) Call Bar_Clark
!  Obtain dimensional temperature
   TD = 100.0*TN
    t = t + dT
!  Write rezult into file = 'Bar_Cl.rez'
    if((t.GE.TMax) .OR. (n.GE.Nmax)) &
   Write(2,102) t,(TD(j),j=1,Jmax)
102Format(' T= ',F5.2,' TD= ',11F7.2)
!
!  If maximal time or number of time-steps exceeded, exit block
   if((t.GE.TMax) .OR. (n.GE.Nmax)) exit
   end do
!
!  Obtain exact solution and compare
   sum = 0.0d0
!
   Call EXACT
   do j = 1,Jmax
           dmp = TE(j) - TD(j)
           sum = sum + dmp*dmp
   end do
   Write(2,103) t,(TE(j),j=1,Jmax)
103Format(/,' T= ',F5.2,' TE= ',11F7.2,//)
!
!  rms is the RMS error
   avs = sum/(1.+ ajm)
   rms = dsqrt(avs)
   Write(2,104) rms
104Format(' RMS DIF = ',1pD11.4,/)
   Close(2)
   Stop

   Contains

   Subroutine FTCS
!  The FTCS procedure
   Integer i
   Real,dimension(Jmax) :: DUM
        DUM = 0.0
   do i = 2,Jmap
    DUM(i) = (1.-2.*s)*TN(i)+s*(TN(i-1)+TN(i+1))
   end do
   do i = 2,Jmap
      TN(i) = DUM(i)
   end do
   End Subroutine FTCS

   Subroutine Duf_Frank
!  The DuFort-Frankel' procedure
   Integer i
   Real,dimension(Jmax) :: DUM
        DUM = 0.0
```

```
   do i = 2,Jmap
     DUM(i) = as*(TN(i-1)+TN(i+1))+bs*TN0(i)
   end do
        TN0 = TN
   do i = 2,Jmap
      TN(i) = DUM(i)
   end do
   End Subroutine Duf_Frank

   Subroutine Bar_Clark
!  The Barakat-Clark' procedure
   Integer i,ii
   Real pp,qq
   Real,dimension(Jmax) :: P,P0,Q,Q0
             P = 0.0 ;  Q = 0.0
   If (n .EQ. 1) Then
            P0 = TN0 ; Q0 = TN0
                  End If
               pp = TN0(1) ;  qq = TN0(Jmax)
             P(1) = pp ; P(Jmax) = qq
         Q(1) = pp ; Q(Jmax) = qq
   do i = 2,Jmap
           ii = Jmax - i + 1
       pp =  as1*P0(i) + as2*(pp+P0(i+1))
           qq = as1*Q0(ii) + as2*(Q0(ii-1)+qq)
           P(i)  = pp ; Q(ii) = qq
   end do
   do i = 2,Jmap
      TN(i) = 0.5*(P(i)+Q(i))
   end do
        P0 = P ; Q0 = Q
   End Subroutine Bar_Clark

   Subroutine EXACT
!  Exact solution of the transient heat conduction
   Integer m,i
   Real ai,dam,am,dxm,dtm
   Real,Dimension(Jmax) :: X0
   TE = 0.0
   Do i = 1,Jmax
            ai = i - 1
     X0(i) = dX*ai
           dxm = Pi*X0(i)/aL
           dtm = -Alph*Pi*Pi*t/aL/aL
!  Limit the argument size of exp(dtm)
           If(dtm .LT. - 75.0) dtm = - 75.0
                   dtm = exp(dtm)
           If(t .LE. 0.0) dtm = 1.0
     TE(i) = Co*dtm*sin(dxm)
   End Do
   End Subroutine EXACT
   End Program Diff
```

The results of computation given in the Table show that increase of temporal step leads to the decrease of accuracy. Moreover the solution becomes unstable: s is also growing up to its limit value (s= 0.5) and exceeds the limit in the third case (s= 0.8).

Схема	Δx	Среднеквадратическая ошибка *rms*		
		$\Delta t = 0.1$	$\Delta t = 0.25$	$\Delta t = 0.4$
		s=0.2	*s=0.5*	*s=0.8*

FTCS	0.1	2.1271D-01	7.4334D-01	2.6887D+27

6. The algorithm is similar to the one applied for the previous task. The listing of the modified FORTRAN-90 program Diff has been given above with comments. The results of numerical solution presented in the Table:

Scheme	Δx	Mean-square error *rms*		
		$\Delta t = 0.1$	$\Delta t = 0.25$	$\Delta t = 0.4$
FTCS	0.1	2.1271D-01	7.4334D-01	2.6887D+27
	0.05	2.6887D+27	1.2565D+27	1.4895D+25
Dufort-Frankel	0.1	8.0171D-02	2.9964D-01	9.8013D-01
	0.05	2.6009D-01	1.9072D+00	5.3075D+00

have shown that double decrease of the spatial grid step by the same time-step does not improve the solution accuracy for the FTCS scheme. Moreover the numerical solution strongly diverges with the exact one that is caused by increase of the parameter s, which is 0.8, 2.0 and 2.4 in the first, second and third case, respectively. The accuracy of DuFort-Frankel scheme has also some tendency to decrease with spatial step decrease by keeping a constant time-step. This is due to the ratio $\Delta t/\Delta x$, which is growing as it is clearly observed from the modified equation (4.13).

7. The algorithm and computer code are the same as in two previous tasks. The results of computations:

Scheme	Δx	Mean-square error *rms*		
		$\Delta t = 0.1$	$\Delta t = 0.25$	$\Delta t = 0.3$
Dufort-Frankel	0.1	8.0171D-02	2.9964D-01	9.8013D-01
	0.05	2.6009D-01	1.9072D+00	5.3075D+00
Barakat-Clark	0.1	1.4665D-01	1.1470D-01	5.1196D-02
	0.05	1.5682D-02	1.2837D-01	4.1522D-01

show that Barakat-Clark scheme has the highest accuracy $\approx O((\Delta t)^2,(\Delta x)^2)$ comparing to both Dufort-Frankel and FTCS scheme (see also the results of the previous task).

8. The results of calculation by Crank-Nicholson scheme using the algorithm (4.19) are given below. The scheme has accuracy $\approx O((\Delta t)^2,(\Delta x)^2)$ in all range of Δt .

Scheme	Δx	Mean-square error *rms*		
		$\Delta t = 0.1$	$\Delta t = 0.25$	$\Delta t = 0.3$
Crank-Nicolson	0.1	1.2378D-01	1.2063D-01	1.1949D-01

	0.05	3.1569D-02	3.0261D-02	2.9813D-02

```
  Program Diff
  !Solves 1D heat conduction with FI, Crank-Nicolson or GI3L scheme
  Implicit none
  !   Jmax - max point number along the rod
  Integer,Parameter :: Jmax = 11
  Integer,Parameter :: Jmap = Jmax - 1
  Integer,Parameter :: Jmaf = Jmax - 2
  !       Nmax - max number of time steps
  Integer,Parameter :: Nmax = 500
  !       MaxEx- max number of terms in the exact solution
  Integer,Parameter :: MaxEx = Jmap
  Integer n,j,Itip,i,jm,kj
  !
  Real,Parameter :: Pi = 3.1415927
  !       Alph - the thermal diffusivity
  Real,Parameter :: Alph = 0.01
  Real,Parameter :: Co   = 100.0 ! grad C
  Real,Parameter :: aL   = 1.0 ! wall thickness
  Real,Parameter :: tMin = 4.5   ! Min Time
  Real,Parameter :: tMax = 12.0  ! Max Time
  !       t        - non-dimensional time
  !       dT   - time-step
  !       dX       - step along the rod
  Real dT,t,s,dX,ajm,ad,bd,cd,Gam,Bet,dd
  Real*8 sum,avs,rms,dmp
  !       TN(*) - non-dimensional temperature on n-th layer
  !   TN0(*)- non-dimensional temperature on (n-1)-th layer
  !       TD(*) - dimensional temperature
  !       TE(*) - dimensional temperature (exact solution)
  Real,Dimension(Jmax) :: TD,TE,TN,TN0
  !       A(1-2,*)  - coefficient behind diagonal
  !       A(3,*)    - coefficient on diagonal
  !       A(4-5,*)  - coefficient ahead diagonal
  Real,Dimension(5,Jmaf) :: A
  !       D(*)   - elements of constant vector
  !       DUM(*) - work array
  Real,Dimension(Jmaf) :: D,DUM
  !       ELX(3) - Differential operator
  Real,Dimension(3) :: ELX = (/1.,-2.,1./)
          External BanFac,BanSol

  !       Input data
       ajm = Jmap
           dX = 1.0/ajm
  !                dT = 0.1
  !          s = dT*Alph/dX/dX
             s = 1.0
  !          s = 1./sqrt(12.0)
                 dT = dX*dX*s/Alph
Write(*,'(/a\)') ' Input Tip (1-CN,2-GI3L,3-GI3Lopt) = '
  Read(*,*) Itip
  !
  Select case (Itip)
  case (1)       ! Initialization Crank-Nicolson
          Gam = 0.0 ; Bet = 0.5
          Open(2,file = 'Dif_CN.rez')
          Write(2,'(/a)') '  Crank-Nicolson scheme'
Write(2,'(/a,F5.3,a,F5.3)') ' Beta =',Bet,' Gama = ',Gam
  case (2)       ! Initialization GI3L
          Gam = 0.5 ; Bet = 0.5 + Gam
          Open(2,file = 'Dif_GI3L.rez')
```

```
          Write(2,'(/a)') '  GI3L scheme'
  Write(2,'(/a,F5.3,a,F5.3)') ' Beta =',Bet,' Gama = ',Gam
    case (3)       ! Initialization GI3Lopt
          Gam = 0.5 ; Bet = 0.5 + Gam + 1./12./s
          Open(2,file = 'Dif_GI3Lopt.rez')
          Write(2,'(/a)') '  GI3Lopt scheme'
  Write(2,'(/a,F5.3,a,F5.3)') ' Beta =',Bet,' Gama = ',Gam
    End Select

    ad = - Bet*s
    bd = 1. + Gam + 2.*Bet*s
    cd = ad
    !
    !       Write input data into file = 'Dif_xx.rez'
    Write(*,100) Jmax,MaxEx,Nmax,TMax
    Write(2,100) Jmax,MaxEx,Nmax,TMax
100 Format(/' Jmax = ',i3,' MaxEx = ',i3,' Nmax = ',i3, &
             ' tMax = ',1pE12.5)
    Write(*,101) s,Alph,dT,dX
    Write(2,101) s,Alph,dT,dX
101 Format(/' s = ',1pE11.4,' Alpha = ',1pE11.4,' dT = ',&
            1pE11.4,' dX = ',1pE11.4,/)

    !   Set the matrix for tridiagonal equation array
    do j = 1,Jmaf
          A(1,j) = 0.0
          A(2,j) = ad
          A(3,j) = bd
          A(4,j) = cd
          A(5,j) = 0.0
    end do
    !       Factorizes the tridiagonal matrix
    Call BanFac(Jmaf,A)
    !       Obtain initial condition from exact solution
    !
    t = tMin - dT
    Do n = 1,2
      t = t + dT
    !
      Call EXACT
    !
      do j = 2,Jmap
           If(n .EQ. 1) TN0(j) = TE(j)/100.0
           If(n .EQ. 2) TN(j)  = TE(j)/100.0
      end do
    End Do
    !       Set boundary conditions
       TN(1) = 0.0 ;    TN0(1) = 0.0
    TN(Jmax) = 0.0 ; TN0(Jmax) = 0.0
    n = 0
    t = tMin
    If(Itip .GE. 2) t = tMin + dT
    !       Start iterations
    do
    n = n + 1
    !       Compute elements of constant vector
    !
    Do j = 2,Jmap
      jm = j - 1
       D(jm) = (1.+2.*Gam)*TN(j)-Gam*TN0(j)
             dd = 0.0
          do i = 1,3
            kj = j - 2 + i
            dd = dd + ELX(i)*TN(kj)
```

```
        end do
        D(jm) = D(jm) + dd*s*(1.-Bet)
End Do
           D(1) = D(1) - A(2,1)*TN(1)
        D(Jmaf) = D(Jmaf) - A(4,Jmaf)*TN(Jmax)
!
!       Compute banded system of equations
Call BanSol(Jmaf,A,D,DUM)
!
Do j = 2,Jmap
   TN0(j) = TN(j)
        TN(j) = DUM(j-1)
End Do
!
!       Obtain dimensional temperature
TD = 100.0*TN
 t = t + dT
!       Write result into file = 'Dif_xx.rez'
    if((t.GE.tMax) .OR. (n.GE.Nmax)) &
Write(2,102) t,(TD(j),j=1,Jmax)
102     Format(' T= ',F5.2,' TD= ',11F7.2)
!
!If maximal time or time-step number exceeded, then exit
if((t.GE.tMax) .OR. (n.GE.Nmax)) exit
end do
!       Obtain exact solution and compare
sum = 0.0d0
!
Call EXACT
!
do j = 1,Jmax
        dmp = TE(j) - TD(j)
        sum = sum + dmp*dmp
end do
Write(2,103) t,(TE(j),j=1,Jmax)
103     Format(/,' T= ',F5.2,' TE= ',11F7.2,//)
!
!       rms is the RMS error
avs = sum/(1.+ ajm)
rms = dsqrt(avs)
Write(2,104) rms
104     Format(' RMS DIF = ',1pD11.4,/)
Close(2)
Stop

Contains

Subroutine EXACT
!     Exact solution for the transient heat conduction problem
Integer m,i
Real ai,dam,am,dxm,dtm
Real,Dimension(Jmax) :: X0

TE = 0.0
Do i = 1,Jmax
         ai = i - 1
  X0(i) = dX*ai
        dxm = Pi*X0(i)/aL
        dtm = -Alph*Pi*Pi*t/aL/aL
!
!       Limit the argument size of exp(dtm)
        If(dtm .LT. - 75.0) dtm = - 75.0
                dtm = exp(dtm)
        If(t .LE. 0.0) dtm = 1.0
```

```
  TE(i) = Co*dtm*sin(dxm)
End Do
End Subroutine EXACT
End Program Diff

Subroutine BanFac(n,B)
!       Factorizes band matrix into L.U
!
Implicit none
Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
Real,Dimension(5,n) :: B
!
np = n - 1
Do j = 1,np
        jp = j + 1
                B(2,jp) = B(2,jp)/B(3,j)
        B(3,jp) = B(3,jp) - B(2,jp)*B(4,j)
End Do
Return
End Subroutine BanFac

Subroutine BanSol(n,B,R,X)
!   Uses L.U factorization to solve tridiagonal system
Implicit none
Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
Real,Dimension(5,n) :: B
!       R(*)  - element of constant vector
!       X(*)  - results
Real,Dimension(n) :: R,X
np = n - 1
Do j = 1,np
        jp = j + 1
        R(jp) = R(jp) - B(2,jp)*R(j)
End Do
X(n) = R(n)/B(3,n)
Do j = 1,np
        jp = N - j
        X(jp) = (R(jp)-B(4,jp)*X(jp+1))/B(3,jp)
End Do
Return
End Subroutine BanSol
```

9. The algorithm (4.20), (4.21) was used. The Thomas's marching method solved the LAEA, which has been obtained at each time-step by discretization. The same computer code Diff was applied. Comparative calculations are presented in the Table:

Scheme GI3L $\gamma = 1.0$	Δx	Mean-square error *rms*		
		$\Delta t = 0.1$	$\Delta t = 0.25$	$\Delta t = 0.3$
$\beta = 1.5$	0.1	1.2127D-01	1.1171D-01	1.0786D-01
	0.05	3.0544D-02	2.4496D-02	2.1794D-02

$\beta=\beta_{OPT}$	0.1	1.1894D-03	7.6146D-03	1.0438D-02
	0.05	1.3993D-03	6.1054D-03	8.5712D-03

Here is $\beta_{OPT}=0.5+\gamma-1/(12s)$. The truncation error of the modified equation is

$$G_j^n = \alpha s\Delta x^2\left[\frac{1}{2}+\gamma-\frac{1}{12s}-\beta\right]\bar{T}_{xxxx} + O\left(\Delta t^2,\Delta x^4\right)\cdot \qquad \text{(K-4.5)}$$

According to (K-4.5), the accuracy is $\approx O((\Delta t)^2,(\Delta x)^2)$ by $\gamma=1.0$, $\beta=1.5$ and $\approx O((\Delta t)^2,(\Delta x)^4)$ by $\beta=0.5+\gamma-1/(12s)$. It has been proved by calculations. As it is seen from the Table in the task 8, this numerical scheme is preferable comparing to the Crank-Nicholson scheme, especially by $\beta=\beta_{OPT}$. Since the main advantages of the three-layer scheme reveal for a rigid boundary problem having the matrix with diagonal preference (see task 1), thus the three-layer schemes with weight coefficients are preferable for more complex CFD problems too.

10. This task and the other optional tasks are proposed to the students for self-training.

Chapter 6

1. The modified equation for (5.16) is obtained using the Taylor series for the function in three node points: T_j^{n+1}, T_{j+1}^n, T_{j-1}^n by Δx and Δt. Thus,

$$T_j^{n+1} = T_j^n + \Delta t \cdot T_t + \frac{(\Delta t)^2}{2} \cdot T_{tt} + \frac{(\Delta t)^3}{6} T_{ttt} + \dots \tag{K-6.1}$$

$$T_{j+1}^n = T_j^n + \Delta x \cdot T_x + \frac{(\Delta x)^2}{2} \cdot T_{xx} + \frac{(\Delta x)^3}{6} T_{xxx} + \dots \tag{K-6.2}$$

$$T_{j-1}^n = T_j^n - \Delta x \cdot T_x + \frac{(\Delta x)^2}{2} \cdot T_{xx} - \frac{(\Delta x)^3}{6} T_{xxx} + \dots \tag{K-6.3}$$

Substitute (K-6.1)-(K-6.3) into (5.16), simplify the similar terms and put to the right the terms with $T_{tt}; T_{ttt}; T_{xx}; T_{xxx}$. This yields

$$T_t + cT_x = -\frac{\Delta t}{2} T_{tt} - \frac{(\Delta t)^2}{6} T_{ttt} + \frac{(\Delta x)^2}{2\Delta t} T_{xx} - c\frac{(\Delta x)^2}{6} \cdot T_{xxx} + \dots \tag{K-6.4}$$

Now replace the temporal derivatives by spatial ones using the scheme proposed in 5.1.2:

$$T_{tt} = c^2 T_{xx} - \frac{\Delta t}{2} T_{ttt} + c\frac{\Delta t}{2} T_{ttx} + \frac{(\Delta x)^2}{2\Delta t} T_{xxt} - c\frac{(\Delta x)^2}{2\Delta t} \cdot T_{xxx} + \dots \tag{K-6.5}$$

$$T_{ttt} = c^2 T_{xxt} + O(\Delta t, (\Delta x)^2), \qquad T_{xxt} = -cT_{xxt} + O(\Delta t, (\Delta x)^2),$$
$$T_{ttx} = c^2 T_{xxx} + O(\Delta t, (\Delta x)^2). \tag{K-6.6}$$

Substituting of (K-6.5), (K-6.6) into (K-6.4) results in the modified equation for the Lax scheme:

$$T_t + cT_x = \frac{c\Delta x}{2}\left(\frac{1}{\nu} - \nu\right) T_{xx} + \frac{c(\Delta x)^2}{3}\left(1 - \nu^2\right) T_{xxx} + \dots \tag{K-6.7}$$

2. To get the modified equation for (5.20), the function is expanded in the points T_j^{n+1}, T_j^{n-1}, T_{j+1}^n, T_{j-1}^n by Δx or Δt using Taylor series:

$$T_j^{n+1} = T_j^n + \Delta t \cdot T_t + \frac{(\Delta t)^2}{2} T_{tt} + \frac{(\Delta t)^3}{6} T_{ttt} + \frac{(\Delta t)^4}{24} T_{tttt} + \frac{(\Delta t)^5}{120} T_{ttttt} + \dots, \tag{K-6.8}$$

$$T_j^{n-1} = T_j^n - \Delta t \cdot T_t + \frac{(\Delta t)^2}{2} T_{tt} - \frac{(\Delta t)^3}{6} T_{ttt} + \frac{(\Delta t)^4}{24} T_{tttt} - \frac{(\Delta t)^5}{120} T_{ttttt} + \dots, \tag{K-6.9}$$

$$T_{j+1}^n = T_j^n + \Delta x \cdot T_x + \frac{(\Delta x)^2}{2} T_{xx} + \frac{(\Delta x)^3}{6} T_{xxx} + \frac{(\Delta x)^4}{24} T_{xxxx} + \frac{(\Delta x)^5}{120} T_{xxxxx} + \dots, \tag{K-6.10}$$

$$T_{j-1}^n = T_j^n - \Delta x \cdot T_x + \frac{(\Delta x)^2}{2} T_{xx} - \frac{(\Delta x)^3}{6} T_{xxx} + \frac{(\Delta x)^4}{24} T_{xxxx} - \frac{(\Delta x)^5}{120} T_{xxxxx} + \dots \tag{K-6.11}$$

Substituting these equations into (5.20) and performing some transformations with similar terms, one can put all the terms containing T_{ttt}, T_{ttttt}, T_{xxx}, T_{xxxxx} to the right:

$$T_t + cT_x = -\frac{(\Delta t)^2}{6}T_{ttt} - \frac{(\Delta t)^4}{120}T_{ttttt} - \frac{c(\Delta x)^2}{6}T_{xxx} - \frac{c(\Delta x)^4}{120}T_{xxxxx} + \ldots \qquad \text{(K-6.12)}$$

Now replace the temporal derivatives by spatial ones using the scheme proposed in 5.1.2:

$$T_{ttt} = -c^3 T_{xxx} - c^2\frac{(\Delta t)^2}{6}T_{tttxx} - c^3\frac{(\Delta x)^2}{6}\cdot T_{xxxxx} - \frac{(\Delta t)^2}{6}T_{ttttt} + \qquad \text{(K-6.13)}$$
$$c\frac{(\Delta t)^2}{6}T_{tttxx} - c^2\frac{(\Delta x)^2}{6}T_{xxxtt} + c^2\frac{(\Delta x)^2}{6}T_{txxxx} + \ldots$$

$$T_{tttxx} = -c^3 T_{xxxxx} + O((\Delta t)^2,(\Delta x)^2), \quad T_{ttttt} = -c^5 T_{xxxxx} + O((\Delta t)^2,(\Delta x)^2),$$
$$T_{ttxxx} = c^2 T_{xxxxx} + O((\Delta t)^2,(\Delta x)^2), \qquad \text{(K-6.14)}$$
$$T_{xxxxt} = -cT_{xxxxx} + O((\Delta t)^2,(\Delta x)^2), \quad T_{tttxx} = c^4 T_{xxxxx} + O((\Delta t)^2,(\Delta x)^2).$$

Substitution of (K-6.13)-(K-6.14) in (K-6.12) results in modified equation for the Skip method ("leap-frog"):

$$T_t + cT_x = \frac{c(\Delta x)^2}{6}\left(\nu^2 - 1\right)T_{xxx} - \frac{c(\Delta x)^4}{120}\left(9\nu^4 - 10\nu^2 + 1\right)T_{xxxxx} + \ldots \qquad \text{(K-6.15)}$$

3. The algorithm (5.16) is used. The calculations are performed untill Nmax exceeded. Numerical solution obtained is compared to the exact solution of the problem:

$$T_E(t,x) = \begin{cases} 0, & 0 \le x \le ut, \\ \sin\left[10\pi(x-ut)\right], & ut \le x \le ut + 0.1, \\ 0, & ut + 0.1 \le x \le 1.0, \end{cases} \qquad \text{(K-6.16)}$$

which is satisfied up to $t=0.9/u$. Both solutions are given in Fig. K-6.1 for $t=8.0$[#].

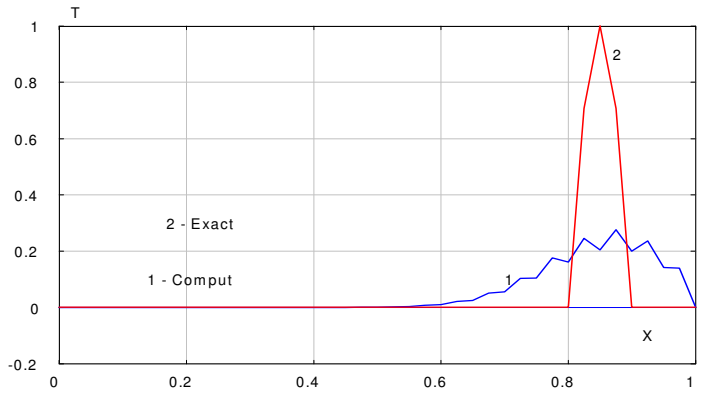


Fig. K-6.1. Solution of the convection equation using the Lax scheme.

Though the mean-square error computed by formula

[#] Here and later on, the shareware WinGnuPlot program was used, which is available on the Net.

$$rms = \left[\left(\sum_{j=1}^{J\max} \left(T_j - TE_j \right)^2 \right) / J\max \right]^{1/2} \qquad \text{(K-6.17)}$$

was small enough (RMS=1.7482D-01), nevertheless the numerical solution is diffusively smoothed similar to the one obtained by introducing the term with artificial viscosity. Listing of the computer code (Fortran-90) is given below with user-friendly comments.

```
Program Wave
!Solves 1D sine-wave equation by explicit and implicit schemes
Implicit none
!       Jmax - max point number along the rod
Integer,Parameter :: Jmax = 41
Integer,Parameter :: Jmap = Jmax - 1
Integer,Parameter :: Jmaf = Jmax - 2
!       Nmax - max number of time steps
Integer,Parameter :: Nmax = 40
!       MaxEx- max number of the terms in the exact solution
Integer,Parameter :: MaxEx = Jmap
Integer n,j,Itip,i,jm,n0,Istop
Real,Parameter :: Pi = 3.141592654
!       Alph - the thermal diffusivity
Real,Parameter ::  U   = 0.1 ! velosity
Real,Parameter :: aL   = 1.0 ! front thickness
!       t    - dimensionless time
!       dT   - temporal step
!       dX   - spatial step
!       C    - Courant number
!       tMin - Min Time
!       tMax - Max Time
Real dT,dX,t,tMin,tMax,atim,ajm,bT
Real aI,bI,cI,aE,bE,cE,delta,C
Real*8 sum,avs,rms,dmp
!       TN(*) - dimensionless temperature on the n-th layer
!      TN0(*)- dimensionless temperature on the     (n-1)-th layer
!       TD(*) - dimensional temperature
!       TE(*) - dimensional temperature (exact solution)
Real,Dimension(Jmax) :: TD,TE,TN,TN0
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
Real,Dimension(5,Jmaf) :: A
!       R(*)   - elements of constant vector
!       DUM(*) - workarrays
Real,Dimension(Jmaf) :: R,DUM
!       ELX(3) - Differential operator
Real,Dimension(3) :: ELX = (/1.,-2.,1./)
        External BanFac,BanSol

!       Input data
           A = 0.0
        atim = Nmax
        tMin = 0.0
         ajm = Jmap
          dX = aL/ajm
!                dT = 0.1
   delta = 0.0
```

```
Do
! Outer cycle

Write(*,'(/a\)')' Input Tip (1-Lax,2-LW,3-ReShuf,4-CTI) = '
Read(*,*) Itip
            Write(*,'(/a\)') ' Input Courant number = '
                Read(*,*) C
                     dT = C*dX/U
       tMax = dT*atim
!
Select case (Itip)
case (1)       ! Initialization Lax scheme
aE = 0.5*(1.+C); bE = 0.0; cE = 0.5*(1.-C)
        Open(2,file = 'Wave_Lax.rez')
        Write(2,'(/a)') '  Lax scheme'
case (2)    ! Initialization Lax-Wendroff scheme
    aE = 0.5*C*(1.+C); bE = 1.-C*C; cE = -0.5*C*(1.-C)
        Open(2,file = 'Wave_LW.rez')
        Write(2,'(/a)') '  Lax-Wendroff scheme'
        case (3)       ! Initialization Reshuffle)
!                        (or lear-frog scheme
        aE = C; bE = 0.0; cE = -C
        Open(2,file = 'Wave_reshuffle.rez')
   Write(2,'(/a)')' Reshuffle (or lear-frog) scheme'
case (4)       ! Initialization Central time
!                       implicit scheme
        aI = -0.25*C; bI = 1.0; cI =  0.25*C
        aE =  0.25*C; bE = 1.0; cE = -0.25*C
        Open(2,file = 'Wave_CTI.rez')
Write(2,'(/a)') '  Central time implicit scheme'
        Write(2,'(/a,F5.3,a,F5.3,a,F5.3)') ' aI = ',aI &
                                  ,' bI = ',bI,' cI = ',cI
!   Set the matrix for tridiagonal equation array
do j = 1,Jmaf
        A(1,j) = 0.0
        A(2,j) = aI
        A(3,j) = bI
        A(4,j) = cI
        A(5,j) = 0.0
end do
!       Factorization of the tridiagonal matrix
Call BanFac(Jmaf,A)
!
End Select
!       Write input data into file = 'Wave_xx.rez'
Write(2,'(/a,F5.3,a,F5.3)')' C = ',C,' U = ',U
        Write(2,'(/a,F5.3,a,F5.3,a,F5.3)') ' aE = ',aE &
                                  ,' bE = ',bE,' cE = ',cE
Write(*,100) Jmax,MaxEx,Nmax,TMax
Write(2,100) Jmax,MaxEx,Nmax,TMax
100     Format(/' Jmax = ',i3,' MaxEx = ',i3,' Nmax = '&
               ,i3,' tMax = ',1pE12.5)
Write(*,101) dT,dX
Write(2,101) dT,dX
101     Format(/' dT = ',1pE11.4,' dX = ',1pE11.4)

!       Obtain initial condition from exact solution
  n = 0

  Call EXACT
!       Set boundary conditions
   TN(1) = 0.0 ;    TN0(1) = 0.0
TN(Jmax) = 0.0 ; TN0(Jmax) = 0.0
t = tMin ; n0 = 1
```

```
If(Itip .EQ. 3) Then
        t = tMin + dT
   n0 = 2
End If
!
!       Marching solution in time ...
do n = n0,Nmax
!
If(Itip .LT. 4)        Then
!       Explicit schemes
!
Do j = 2,Jmap
        bT = aE*TN(j-1)+bE*TN(j)+cE*TN(j+1)
        If(Itip .EQ. 3) bT = bT + TN0(j)
        DUM(j-1) = bT
End Do
        Else
!       Implicit schemes
!       Compute elements of constant vector
Do j = 2,Jmap
  jm = j - 1
   R(jm) = aE*TN(j-1)+bE*TN(j)+cE*TN(j+1)
End Do
           R(1) = R(1) - A(2,1)*TN(1)
        R(Jmaf) = R(Jmaf) - A(4,Jmaf)*TN(Jmax)
!
!       Compute banded system of equations
Call BanSol(Jmaf,A,R,DUM)
        End If
Do j = 2,Jmap
   TN0(j) = TN(j)
        TN(j) = DUM(j-1)
End Do

!       Obtain dimensional temperature
TD = 1.0*TN
 t = t + dT
!       Write rezult into file = 'Dif_xx.rez'
        if((t.GE.tMax) .OR. (n.GE.Nmax)) Then
        Write(2,'(/a,F5.2)') ' T= ',t
        Write(2,102) (TD(j),j=1,Jmax)
End If
102     Format(' TD= ',11F7.2)
!
!       End time cycle
end do
!
!       Obtain exact solution and compare
Call EXACT
Write(2,'(/a,F5.2)') ' Tmax = ',t
Write(2,103) (TE(j),j=1,Jmax)
103     Format(' TE= ',11F7.2)
!       rms is the RMS error
sum = 0.0d0
!
do j = 1,Jmax
        dmp = TE(j) - TD(j)
        sum = sum + dmp*dmp
end do
avs = sum/(1.+ ajm)
rms = dsqrt(avs)
Write(2,104) rms
104     Format(/,' RMS DIF = ',1pD11.4,/)
Write(*,'(/a\)')' Input Istop (0-Stop,>1-Continue) = '
```

```
Read(*,*) Istop
If(Istop .LT. 1) Exit
End Do
! End outer cycle
Close(2)
Stop

Contains
Subroutine EXACT
!       Exact solution of the transient heat conduction problem
Integer i,ip,Jm,inc
Real ai,dxm
Real,Dimension(Jmax) :: X0
         TE = 0.0 ; TN = 0.0
 Jm = 0.1001/dX + 1.0
inc = U*tMax/dX + 0.001
Do i = 1,Jm
         ai = i - 1
  X0(i) = dX*ai
       dxm = 10.0*Pi*X0(i)/aL
  TN(i) = sin(dxm)
         ip = i + inc
 TE(ip) = TN(i)
 If((Itip .EQ. 3) .AND. (n .LE. 1)) Then
            TN0(i) = TN(i)
                  TN(i+1) = TN0(i)
 End If
       End Do
End Subroutine EXACT
End Program Wave

Subroutine BanFac(n,B)
!       Factorization of the band matrix into L.U
Implicit none
Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
Real,Dimension(5,n) :: B
np = n - 1
Do j = 1,np
       jp = j + 1
       B(2,jp) = B(2,jp)/B(3,j)
       B(3,jp) = B(3,jp) - B(2,jp)*B(4,j)
End Do
Return
End Subroutine BanFac

Subroutine BanSol(n,B,R,X)
!   Uses L.U factorization to solve tridiagonal system
Implicit none
Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
Real,Dimension(5,n) :: B
!       R(*)  - element of constant vector
!       X(*)  - results
Real,Dimension(n) :: R,X
np = n - 1
Do j = 1,np
       jp = j + 1
       R(jp) = R(jp) - B(2,jp)*R(j)
End Do
```

```
X(n) = R(n)/B(3,n)
Do j = 1,np
        jp = N - j
        X(jp) = (R(jp)-B(4,jp)*X(jp+1))/B(3,jp)
End Do
Return
End Subroutine BanSol
```

4. Calculations were done with algorithm (5.24) and the same computer code as in previous task. The results by Lax-Wendroff scheme and exact solution are in Fig. K-6.2.

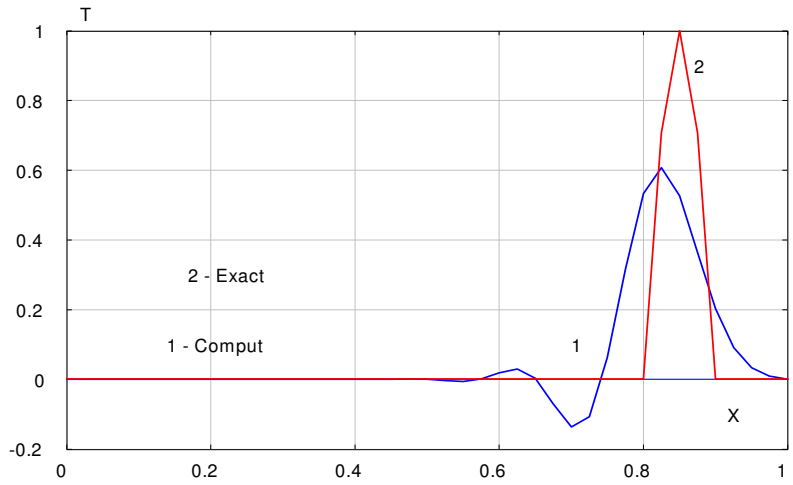


Fig. K-6.2. Solution of the convection equation with Lax-Wendroff scheme.

Accuracy of the numerical solution has increased comparing to the previous task (RMS=0.4200D-1) due to the scheme of a second order accuracy. But this numerical solution has primary wave with decreased amplitude and secondary waves in a trace of the primary wave. The wave packet, following the base solution, is called “dispersive trace”.

5. The same computer code was applied using the algorithm (5.20). The results of computation with Skip scheme (Leapfrog) and the exact solution are given in Fig. K-6.3.

The second-order accuracy numerical scheme applied here, as previously, gave the result, which is worse than that obtained by the Lax-Wendroff second order scheme. And what is surprising, the result is even worse than the one with the Lax scheme (RMS=2.5678D-01). This is due to an absence of the even order derivatives in the modified equation (K-6.15) for this scheme, which are responsible for the dissipative properties. Thus, the numerical scheme has too strong dispersion.

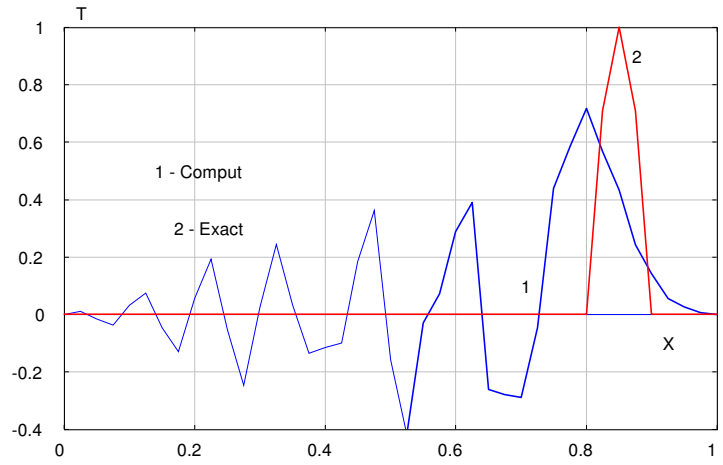


Fig. K-6.3. Solution of the convection equation with skip ("leap-frog") scheme.

6. The calculations were performed by algorithm (5.30) with the same computer code. The linear algebraic equation array has been solved at each time-step using Thomas' marching method. The results of computation using the centered by time implicit (CTI) numerical scheme are given in Fig. K-6.4. The above-mentioned peculiarity is also here due to an absence of the even order derivatives in the modified equation:

$$T_t + cT_x = -\frac{c}{2}\left[\frac{c^2(\Delta t)^2}{6} + (\Delta x)^2\right]T_{xxx} - \frac{c}{8}\left[\frac{(\Delta x)^4}{15} + c^2\frac{(\Delta x)^2(\Delta t)^2}{3} + c^3\frac{(\Delta t)^4}{10}\right]T_{xxxxx} + \ldots \qquad \text{(K-6.18)}$$

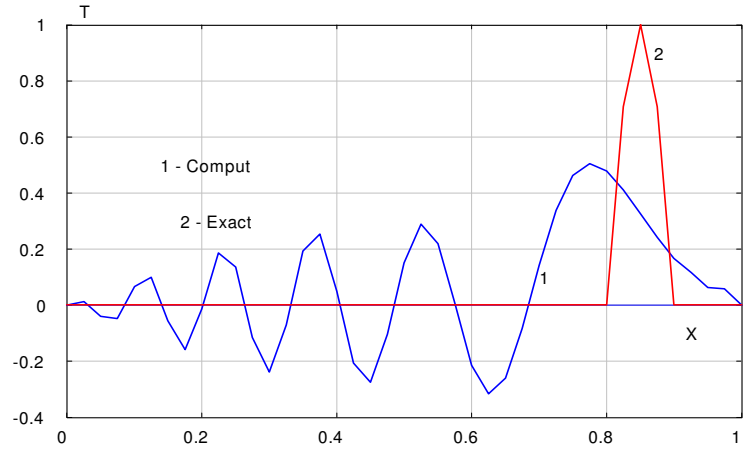


Fig. K-6.4. Solution of the convection equation with CTI scheme.

7. The algorithm (6.15) is used. The calculations are performed untill Nmax exceeded. Numerical solution obtained is compared to the exact solution of the problem:

$$T_E(t,x) = 0.5 - \frac{2}{\pi}\sum_{m=1}^{MaxEx}\sin\left[(2m-1)\pi\frac{x-ut}{L}\right]\frac{e^{-\alpha(2m-1)\pi^2 t/L^2}}{2k-1}. \qquad \text{(K-6.19)}$$

Both the exact and numerical solution by $t=t_{max}$ for a few grid Reynolds numbers $\boldsymbol{R_c}$ = 1, 1.5, 2, 3 (corresponding to the Courant numbers) are presented in Fig. K-6.5. It is shown that the numerical solution coincides with the exact one by $\boldsymbol{R_c}$ = 1 while by $\boldsymbol{R_c}$ = 1.5, 2 the difference is remarkable. Nevertheless these solutions are also attainable due to their smooth behaviour and small general inaccuracy (RMS=1.6493D-02, 3.4269D-02). But by $\boldsymbol{R_c}$ >2 the numerical solution becomes unattainable (RMS=1.0992D-01). The listing of the computer code with user-friendly comments is given below.

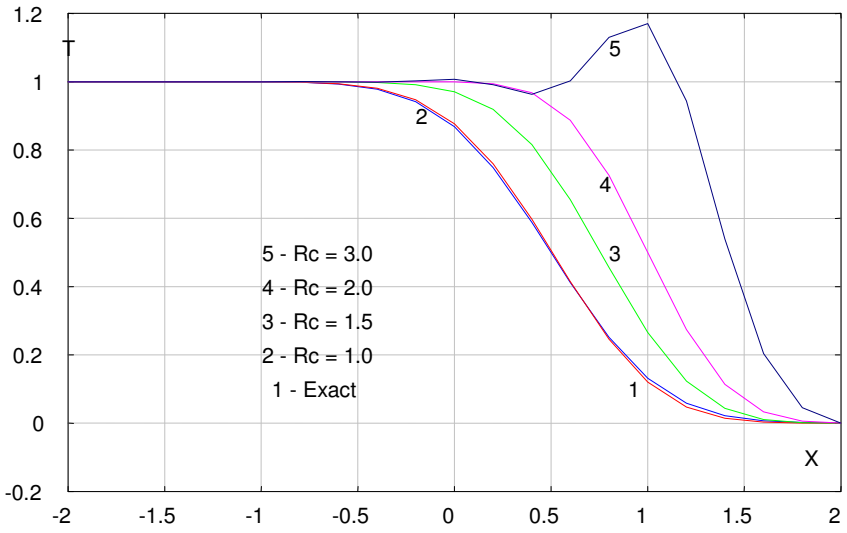


Fig. K-6.5. The solution of one-dimensional transport equation with FTCS scheme.

```
Program Wave
!       Solves 1D propagating temperature-wave equation
!       using various explicit and implicit schemes
        Implicit none
!       Jmax - max point number along the front
        Integer,Parameter :: Jmax = 21
        Integer,Parameter :: Jmap = Jmax - 1
        Integer,Parameter :: Jmaf = Jmax - 2
!       lN   - max number of member series of exact solution
        Integer,Parameter :: lN   = 20
!       Nmax - max number of time steps
        Integer,Parameter :: Nmax = 10
!       MaxEx- max number of terms in the exact solution
```

```
        Integer,Parameter :: MaxEx = 100
        Integer n,j,Itip,i,jm,Istop,mQ
        Real,Parameter :: Pi = 3.141592654
!       Alph - the thermal diffusivity
        Real,Parameter ::  U   = 0.5 ! velosity
        Real,Parameter :: aL   = 4.0 ! front thickness
        Real,Parameter ::  s   = 0.25 ! parameter
!       t    - dimensionless time
!       dT   - temporal step
!       dX   - spatial step
!       C    - Courant number
!       tMin - Min Time
!       tMax - Max Time
!       s    - parameter
!       Alph - parameter
!       Rcel - Grid Reynolds number
!       q    - parameter for 4pt upwind schemes
        Real dT,dX,t,tMin,tMax,atim,ajm,bT,Alph,q,dim,dom
        Real aI,bI,cI,aE,bE,cE,delta,C,qq,Rcel,eL,ss
        Real*8 sum,avs,rms,dmp
!       TN(*) - dimensionless temperature on n-th layer
!       TD(*) - dimensional temperature
!       TE(*) - dimensional temperature (exact solution)
        Real,Dimension(Jmax) :: TD,TE,TN
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
        Real,Dimension(5,Jmaf) :: A
!       R(*)   - elements of constant vector
!       DUM(*) - workarrays
        Real,Dimension(Jmaf) :: R,DUM
!       ELX(3) - Differential operator
        Real,Dimension(3) :: ELX = (/1.,-2.,1./)
        External BanFac,BanSol
!       Input data
                  A = 0.0
               atim = Nmax
               tMin = 0.0
                ajm = Jmap
                 dX = aL/ajm
                 el = lN
        Do
! Outer cycle
        Write(*,'(/a\)') ' Input Tip (1-FTCS,2-LW,3-Ex4Up,4-GCN) = '
        Read(*,*) Itip
            Write(*,'(/a\)') ' Input C = '
                Read(*,*) C
                        q = 0.15 ; delta = 0.0
                        If(Itip .EQ. 4) Then
                         Write(*,'(/a\)') ' Input q,delta = '
                                Read(*,*) q,delta
                        End If
                         mQ = 0
                         ss = s
                         dT = C*dX/U
               tMax = dT*atim
                   Alph = s*dX*dX/dT
        If(Alph .LT. 1.0e-10) Alph = 1.0e-10
        If(Itip .GT. 3 .AND. q .GT. 0.0) q = 0.5 + C*C/4.
                   Rcel = C/s
                     qq = q*C/3.
        If(Itip .LT. 3 .OR. delta .GT. 0.0) qq = 0.0
 If(Itip .GT. 3 .AND. delta .GT. 0.0) delta = 1./6. + C*C/12.
        If(abs(qq) .GT. 0.0001) mQ = 1
```

```
        Select case (Itip)
        case (1)
! Initialization FTCS scheme
                         aE = (0.5*C+ss) + 3.0*qq
                         bE = 1.0 - 2.0*ss - 3.0*qq
                         cE = -0.5*C + ss + qq
                             Open(2,file = 'WaveT_FTCS.rez')
                        Write(2,'(/a)') '  FTCS scheme'
        case (2)
! Initialization Lax-Wendroff scheme
                         ss = s + 0.5*C*C
                         aE = (0.5*C+ss) + 3.0*qq
                         bE = 1.0 - 2.0*ss - 3.0*qq
                         cE = -0.5*C + ss + qq
                                Open(2,file = 'WaveT_LW.rez')
                      Write(2,'(/a)') '  Lax-Wendroff scheme'
        case (3)
! Initialization Explicit 4pt upwind scheme
                         aE = (0.5*C+ss) + 3.0*qq
                         bE = 1.0 - 2.0*ss - 3.0*qq
                         cE = -0.5*C + ss + qq
                             Open(2,file = 'WaveT_Ex4Up.rez')
              Write(2,'(/a)') '  Explicit 4pt upwind scheme'
        case (4)
! Initialization General C-Nicolson scheme
                        aI= delta - 0.25*C - 0.5*s - 1.5*qq
                        bI = 1.0 - 2.0*delta + s + 1.5*qq
                        cI = delta + 0.25*C - 0.5*s - 0.5*qq
                        aE = delta + 0.25*C + 1.5*qq + 0.5*s
                        bE = 1.0 - 2.0*delta - 1.5*qq - s
                        cE = delta - 0.25*C + 0.5*qq + 0.5*s
                          Open(2,file = 'WaveT_GCN.rez')
          Write(2,'(/a)') '  General Crank-Nikolson scheme'
          Write(2,'(/a,F5.3,a,F5.3,a,F5.3)') ' aI = ',aI    &
                                ,' bI = ',bI,' cI = ',cI
        End Select
!
!       Write input data into file = 'WaveT_xx.rez'
        Write(2,'(/, 4(a,F5.3))') ' C = ',C,' U = ',U,      &
                                  ' Q = ',q,' QQ = ',qq
        if(abs(qq) .GT. 0.0001) mQ = 1
        Write(2,'(/a,F5.3,a,F5.3,a,F5.3)') ' aE = ',aE      &
                         ,' bE = ',bE,' cE = ',cE
        Write(*,100) Jmax,MaxEx,Nmax,TMax
        Write(2,100) Jmax,MaxEx,Nmax,TMax
100     Format(/' Jmax = ',i3,' MaxEx = ',i3,' Nmax = ',i3, &
            ' tMax = ',1pE12.5)
        Write(*,101) s,Alph,dT,dX,Rcel
        Write(2,101) s,Alph,dT,dX,Rcel
101    Format(/' s = ',1pE11.4,' Alpha = ',1pE11.4,' dT = ',&
           1pE11.4,' dX = ',1pE11.4,' Rcel = ',1pE11.4,/)

!       Obtain initial condition from exact solution
         n = 0
!
          Call EXACT
!
!       Set boundary conditions
           TN(1) = 1.0 ; TN(Jmax) = 0.0
         t = tMin
!
!       Marching solution in time ...
        do n = 1,Nmax
        If(Itip .LT. 4) Then
```

```
!       Explicit schemes
!
        Do j = 2,Jmap
                bT = aE*TN(j-1)+bE*TN(j)+cE*TN(j+1)
                DUM(j-1) = bT
        End Do
                          Else
!       Implicit schemes
!
!       Set the matrix for tridiagonal system of equations
        If(mQ .EQ. 1) dim = 1.0
        Do j = 2,Jmap
          jm = j - 1
                A(1,jm) = 0.5*qq
                A(2,jm) = aI
                A(3,jm) = bI
                A(4,jm) = cI
                A(5,jm) = 0.0
!       Compute elements of constant vector
              R(jm) = aE*TN(j-1)+bE*TN(j)+cE*TN(j+1)
          If(mQ .EQ. 1) Then
                If(j .GT. 2) dim = TN(j-2)
                  R(jm) = R(jm) - 0.5*qq*dim
          End If
        End Do
                 R(1) = R(1) - A(2,1)*TN(1)
        If(mQ .GT. 0) Then
                 R(1) = R(1) - A(1,1)*TN(1)
                 R(2) = R(2) - A(1,2)*TN(1)
!
!       Reduce matrix A to a tridiagonal form
        Do j = 3,Jmaf
                    jm = j - 1
                   dom = A(1,j)/A(2,jm)
                A(2,j) = A(2,j) - A(3,jm)*dom
                A(3,j) = A(3,j) - A(4,jm)*dom
                A(1,j) = 0.0
                R(j) = R(j) - R(jm)*dom
        End Do
        A(1,1) = 0.0 ; A(1,2) = 0.0
                       End If
        A(2,1) = 0.0
        A(4,Jmaf) = 0.0
!
!       Factorization of the tridiagonal matrix
        Call BanFac(Jmaf,A)
!
!       Computation of the banded system of equations
        Call BanSol(Jmaf,A,R,DUM)
             End If
        Do j = 2,Jmap
                 TN(j) = DUM(j-1)
        End Do
!       Obtain dimensional temperature
        TD = 1.0*TN
         t = t + dT
!       Write rezult into file = 'Dif_xx.rez'
        if((t.GE.tMax) .OR. (n.GE.Nmax)) Then
               Write(2,'(/a,F5.2)') ' T= ',t
                   Write(2,102) (TD(j),j=1,Jmax)
        End If
 102     Format(' TD= ',11F7.2)
!       End time cycle
        end do
```

```
!       Obtain exact solution and compare
!
        Call EXACT
!
        Write(2,'(/a,F5.2)') ' Tmax = ',t
        Write(2,103) (TE(j),j=1,Jmax)
 103     Format(' TE= ',11F7.2)
!       rms is the RMS error
         sum = 0.0d0
        do j = 1,Jmax
                dmp = TE(j) - TD(j)
                sum = sum + dmp*dmp
        end do
        avs = sum/(1.+ ajm)
        rms = dsqrt(avs)
        Write(2,104) rms
 104     Format(/,' RMS DIF = ',1pD11.4,/)
        Write(*,'(/a\)') ' Input Istop (0-Stop,>1-Continue) = '
        Read(*,*) Istop
        If(Istop .LT. 1) Exit
        End Do
! End outer cycle

        Close(2)
        Stop

        Contains

        Subroutine EXACT
!       Exact solution for propagating temperature front
        Integer i,k,ip,Jm,inc
        Real ai,dxm,dem,dam,sne
        Real,Dimension(Jmax) :: X0
         TE = 0.0 ; TN = 0.0
         TE(1) = 1.0 ; TN(1) = 1.0
        Do i = 1,Jmax
                 ai = i - 1
          X0(i) = -2.0 + dX*ai
        End Do
        Do i = 2,Jmap
          If(X0(i) .LT. 0.0) TN(i) = 1.0
          If(abs(X0(i)) .LT. 1.0e-4) TN(i) = 0.5
            dem = 0.0
          Do k = 1,MaxEx
                 ai = 2*k - 1
                dam = ai*Pi/el
                sne = sin(dam*(X0(i)-U*tMax))
                dxm =- Alph*dam*dam*tMax
                If(dxm .LT. -20.0) exit
                dem = dem + (sne/ai)*exp(dxm)
          End Do

          TE(i) = 0.5 - 2.0*dem/Pi

        End Do
        End Subroutine EXACT
        End Program Wave

        Subroutine BanFac(n,B)
!       Factorization of the band matrix into L.U
        Implicit none
        Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
```

```
!       A(4-5,*)  - coefficient ahead diagonal
        Real,Dimension(5,n) :: B
        np = n - 1
        Do j = 1,np
                jp = j + 1
                B(2,jp) = B(2,jp)/B(3,j)
                B(3,jp) = B(3,jp) - B(2,jp)*B(4,j)
        End Do
        Return
        End Subroutine BanFac

        Subroutine BanSol(n,B,R,X)
!       Uses L.U factorization to solve the tridiagonal system
        Implicit none
        Integer n,j,np,jp
!       A(1-2,*)  - coefficient behind diagonal
!       A(3,*)    - coefficient on diagonal
!       A(4-5,*)  - coefficient ahead diagonal
        Real,Dimension(5,n) :: B
!       R(*)  - element of constant vector
!       X(*)  - results
        Real,Dimension(n) :: R,X
        np = n - 1
        Do j = 1,np
                jp = j + 1
                R(jp) = R(jp) - B(2,jp)*R(j)
        End Do
        X(n) = R(n)/B(3,n)
        Do j = 1,np
                jp = N - j
                X(jp) = (R(jp)-B(4,jp)*X(jp+1))/B(3,jp)
        End Do
        Return
        End Subroutine BanSol
```

8. Algorithm (6.27) is used with the same computer code. Numerical solution by the Lax-Wendroff scheme and exact solution of the problem are presented in Fig. K-6.6. Note that numerical solution is very close in this case to the ones obtained with FTCS scheme though the Lax-Wendroff scheme is of the second order. It can be explained, as the last one is FTCS scheme with modified diffusion. In Fig. K-6.7 the exact solution is presented with the numerical one for the explicit four-point upwind scheme

$$\frac{\Delta T_j^{n+1}}{\Delta t} + uL_x^{(4)}T_j^n - \alpha L_{xx}T_j^n = 0\,, \tag{K-6.20}$$

where is

$$L_{xx} = \frac{1}{(\Delta x)^2}\{1,2,1\}; \quad \Delta T_j^{n+1} = T_j^{n+1} - T_j^n$$

and by $u \geq 0$

$$L_x^{(4)}T \equiv \frac{T_{j+1} - T_{j-1}}{2\Delta x} + \frac{q\left(T_{j-2} - 3T_{j-1} + 3T_j - T_{j-1}\right)}{3\Delta x} + 0\left((\Delta x)^2\right).$$

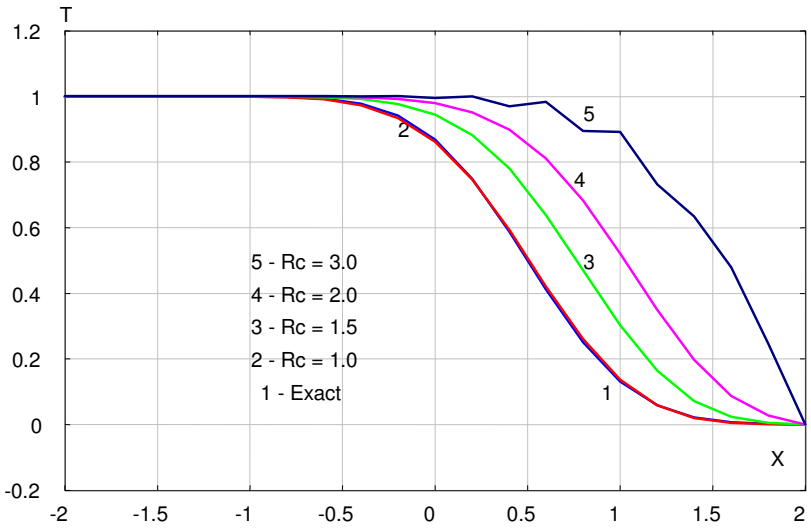


Fig. K-6.6. Solution of 1-D Transport equation with Lax-Wendroff scheme.

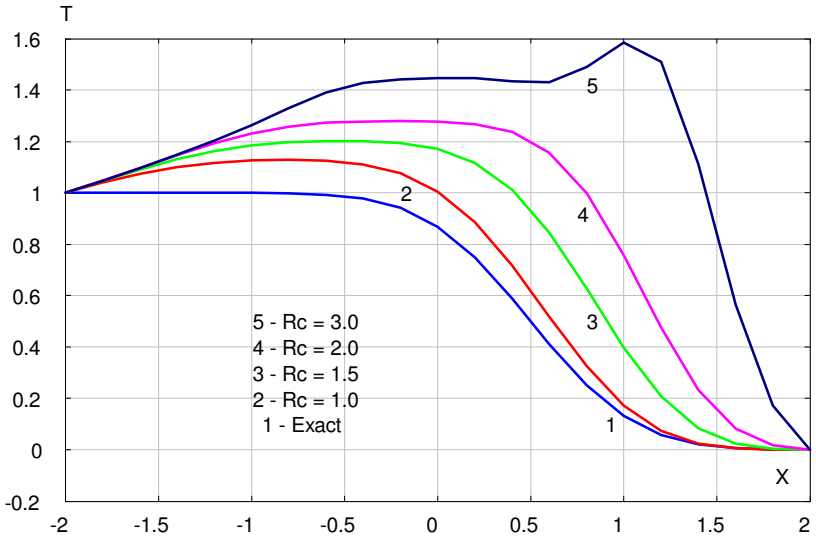


Fig. K-6.7. Solution of 1-D Transport equation with explicit 4-point upwind scheme.

By *q=0.5,* this scheme has the third order of accuracy for the convection term. But the whole accuracy is not sufficient. The results in Fig.K-6.7 correspond to *q=0.15,* by small Courant number they are attainable. A decrease of q increases accuracy. By $\mathbf{R}_c \leq 2.0$, the Lax-Wendroff scheme is of highest accuracy among the explicit schemes.

9. Here the same computer code, algorithm (6.29) and Thomas's marching procedure for the solution of LAEA at each time-step were used. The Crank-Nicolson (CN) scheme has been realized for $q = 0.0$ and $\delta = 0.0$. The results given in Fig. K-6.8 show that this scheme is working well up to $\mathbf{R}_c \le 3.0$. For $\mathbf{R}_c > 3.0$, the additional oscillations make the solution unattainable.

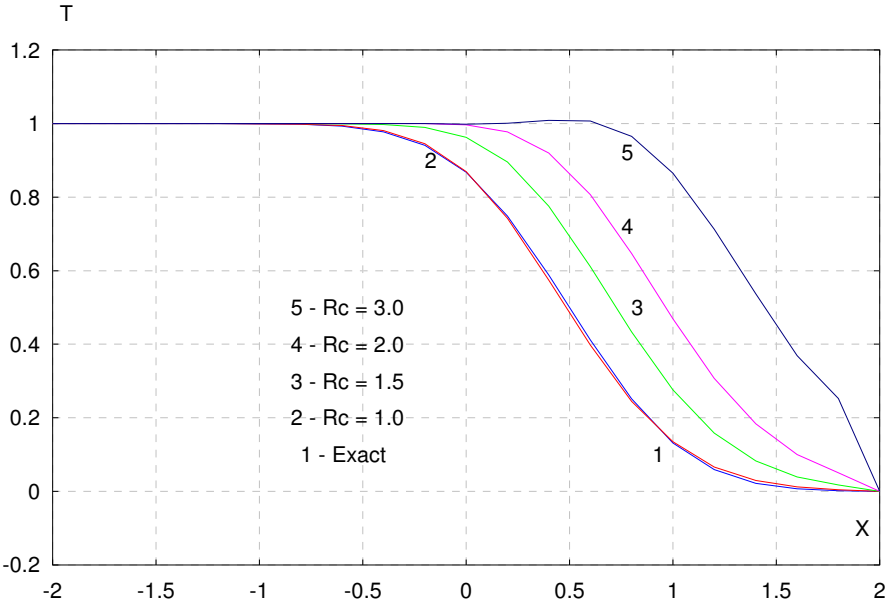


Fig. K-6.8. Solution of 1-D transport equation with implicit Crank-Nicolson scheme.

10. Use algorithms (6.33), (6.35), (6.36). Eq. (6.36) substituted in (6.35), results in a tetradiagonal LAEA at each time-step. The generalized Thomas' algorithm (2.3.1) is used for solution of LAEA. In the above-mentioned computer code Wave, generalization of the Thomas' algorithm is achieved by additional forward marching, which reduces the matrix to the tridiagonal form. The Crank-Nicolson scheme with weights (CNw) has been realized by $q=0.0$ and $\delta=1.6+C^2/12$. In the four-point Crank-Nicolson upwind scheme (4CNup) the following parameters were used: $\delta=0.0$, $q=0.5+0.25C^2$. Crank-Nicolson scheme (CN) is the same as in the previous task. The comparative results obtained are given in the Table:

Scheme	Mean-square error *rms*			
	C = 0.25	*C = 0.375*	*C = 0.5*	*C = 0.75*
CN	5.6237D-03	8.5369D-03	1.1623D-02	1.8821D-02
4CNup	1.8192D-03	2.2561D-03	2.6195D-03	2.4535D-03
CNw	3.5202D-03	3.2737D-03	3.0078D-03	8.9379D-03

Chapter 7

1. The standard mathematical and graphical libraries Compaq Visual FORTRAN (CVF) are used in this chapter. The procedure MOLCH is applied for the solution of the boundary problems by the method of lines[#] for the PDE of the general form:

$$u_t^k = f(x,t,u_1,\ldots,u_m,u_x^1,\ldots,u_x^m,u_{xx}^1,\ldots,u_{xx}^m), k=\overline{1,m} \qquad \text{(K-7.1)}$$

with the initial

$$t = t_0, \qquad u_k = u_k(x,t) \qquad \text{(K-7.2)}$$

and boundary conditions:

$$\alpha_k u_k(t,x_l) + \beta_k u_x^k(t,x_l) = \gamma_k, l=1,n; k=\overline{1,m}. \qquad \text{(K-7.3)}$$

Here $x_1 = a$, $x_n = b$ determine the boundary points of the integration domain by x.

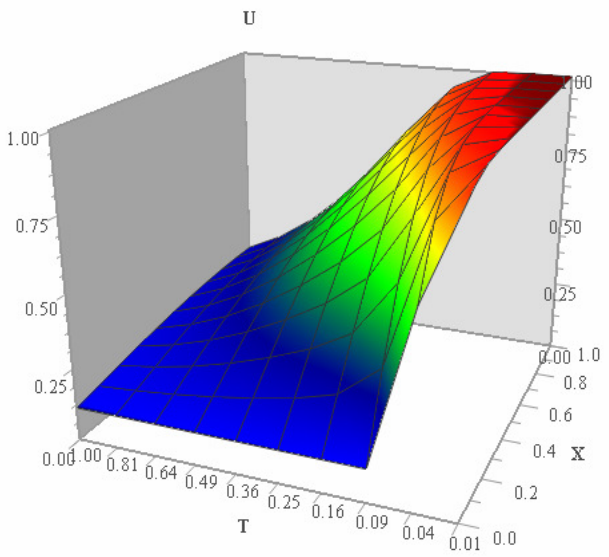


Fig. K-7.1. Solution $u=u(x,t)$ plotted with AV.

According to the algorithm, the PDE is reduced to the ordinary differential equation array through cubic Hermit splines by x, with further application of the collocation method in the Gauss points. Thus, the solution is represented by splines instead of the finite-difference approximation of PDE. With the boundary conditions of the present task, the procedure MOLCH requires to state the values of the derivative for boundary function γ at the points

[#] The algorithm is given in the library IMSL for Compaq Visual FORTRAN 6.0-6.5.

$x_1 = 0$, $x_n = 1$. Function γ changes smoothly by $x=0$ from u_0 (by $t=t_0$) to u_1 (by $t=t_0$). The derivative γ' is calculated in the process of estimation for the cubic spline interpolation, which is obtained with the function CSDER from the library IMSL. The interpolation is done at the beginning of subroutine FCNBC. The function and derivative values $\gamma(t_0)=u_0$, $\gamma(t_{end})=u_1$, $\gamma'(t_0)=0$, $\gamma'(t_{end})=0$ are required by call of the procedure C2HER, which calculates the coefficients for CSDER.

The solution obtained is presented with Array Visualization (AV). AV is included in Compaq Visual FORTRAN 6.5. The main AV subroutine used by plot of the Fig. K-7.1 is Aviewer.exe available shareware at http://www.compaq.com/fortran/. The listing of the Fortran-90 code with comments is given below.

```
        Program Line
!
!     The normalized linear diffusion PDE, Ut=Uxx (0<=x<=1, t > t0) is
!     solved. The initial values are to=0, U(x,to)=Uo=1; "zero-flux"
!     boundary condition at x=1, Ux(1,t)=0(t>to). U(0,t) is abruptly
!     changed from Uo to U1=0.1. This is completed by t=td=0.09.
!     Due to restrictions in the type of boundary conditions sucessfully
!     processed by MOLCH, it is necessary to provide the derivative
!     boundary value function g' at x=0 and at x = 1.
!     The function g at x=0 makes a smooth transition from Uo at t=to to
!     U1 at t=td. Transition phase for g' is computed by evaluating a
!     cubic interpolating polynomial with subprogram CSDER . The
!     interpolation is performed as a first step in the user-supplied
!     routine FCNBC. The function and derivative values: g(to)=Uo,
!     g'(to)=0, g(td)=U1, and g'(td)=0, are used as input to routine
!     C2HER, to obtain the coefficients evaluated by CSDER.
!     Notice: g'(t)=0, t>td. The routine CSDER does not yield this
!     value, thus logic in the routine FCNBC assigns g'(t)=0, t>td.

!     Use DFIMSL
!
      Implicit none
!
!             SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER,PARAMETER :: NPDES = 1, NX = 11, LDY = NPDES, &
                           NSTEP = 10
!               SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER I, IDO, J
      REAL HINIT,T, TEND,TOL,U0, XBREAK(NX), Y(LDY,NX), &
        TSTEP(NSTEP)
!       If you want to start array visualizer (AV) to put
!       AV_true = .True.
        Logical :: AV_true = .False.
!               Array for Array Viewer
        Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!               SPECIFICATIONS FOR FUNCTIONS
        EXTERNAL FCNBC, FCNUT
!
!   Open output file 'Line_01.rez'
        Open(2,file = 'Line_01.rez')
!
!               Set breakpoints and initial conditions
        U0 = 1.0
        Do I=1, NX
```

```
                XBREAK(I) = FLOAT(I-1)/(NX-1)
                Y(1,I) = U0
        End Do
!               Set parameters for MOLCH
        TOL = SQRT(AMACH(4))
        HINIT = 0.01*TOL
        T = 0.0
        IDO = 1
!
        allocate(UXTEND(NSTEP,NX))
!
        Do j= 1,NSTEP
!    Start Main cycle
!
        TEND = FLOAT(J)/FLOAT(NSTEP)
!
!    This puts more output for small t values where action is fastest.
!
        TEND = TEND*TEND
!
!               Solve the problem
        Call MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX,  &
     &                  XBREAK, TOL,HINIT, Y, LDY)
        UXTEND(j,:) = Y(1,1:NX)
           TSTEP(j) = T
!               Print results
        If (J .EQ. 1) Then
        Write(2,'(/a)') '#    T          Y '
        Write(2,'(9x,11F9.2)') (XBREAK(i),i=1,Nx)
        End If
    Write(2,'(F9.2,11F9.4)') T,(Y(1,i),i=1,Nx)
!               Final call to release workspace
        If (J .EQ. NSTEP) IDO = 3
!
        End Do
! End Main cycle
!
!       To use array visualizer
        If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
        deallocate(UXTEND)
!
        Close(2)
        Stop
        End Program Line
!
        SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!
        Implicit none
!
!               SPECIFICATIONS FOR ARGUMENTS
        INTEGER NPDES
        REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!               Define the PDE
        UT(1) = UXX(1)
        RETURN
        END SUBROUTINE FCNUT
!
        SUBROUTINE FCNBC(NPDES, X, T, ALPHA, BETA, GAMP)
!
        Implicit none
!
!               SPECIFICATIONS FOR ARGUMENTS
        INTEGER NPDES
```

```
        REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!               SPECIFICATIONS FOR PARAMETERS
        REAL,PARAMETER :: TDELTA = 0.09, U0 = 1.0, U1 = 0.1
!               SPECIFICATIONS FOR LOCAL VARIABLES
        INTEGER IWK(2), NDATA
        REAL DFDATA(2), FDATA(2), XDATA(2)
!               SPECIFICATIONS FOR SAVE VARIABLES
        REAL BREAK(2), CSCOEF(4,2)
        LOGICAL FIRST
        SAVE BREAK, CSCOEF, FIRST
!               SPECIFICATIONS FOR SUBROUTINES
        EXTERNAL C2HER
!               SPECIFICATIONS FOR FUNCTIONS
        EXTERNAL CSDER
        REAL CSDER
!
        DATA FIRST/.TRUE./
!
        IF (FIRST) Call prep()
!
!               Define the boundary conditions
        IF (X .EQ. 0.0) THEN
!               These are for x=0.
        ALPHA(1) = 1.0
         BETA(1) = 0.0
         GAMP(1) = 0.0
!
! If in the boundary layer, compute non-zero gamma prime.
        IF (T .LE. TDELTA) GAMP(1) = CSDER(1,T,1,BREAK,CSCOEF)
                                        ELSE
!                               These are for x = 1.
        ALPHA(1) = 0.0
         BETA(1) = 1.0
         GAMP(1) = 0.0
                        END IF
!
        Contains
!
        Subroutine prep()
!               Compute the boundary layer data.
            NDATA = 2
         XDATA(1) = 0.0
         XDATA(2) = TDELTA
         FDATA(1) = U0
         FDATA(2) = U1
        DFDATA(1) = 0.0
        DFDATA(2) = 0.0
!               Do Hermite cubic interpolation.
        CALL C2HER (NDATA, XDATA, FDATA, DFDATA, BREAK, &
                    CSCOEF, IWK)
            FIRST = .FALSE.
        End Subroutine prep

        END SUBROUTINE FCNBC

        Subroutine v3D(fun,X,T,mValues,nValues)
!               To output array as vector graph
        Use avdef
        Use avviewer
        Use dflib
        Integer :: nValues,mValues
        Real    :: fun(mValues,nValues),X(nValues),T(mValues)
        Integer :: hv,status,nError
        Character(1) :: key
```

```
        Character(av_max_label_len) :: xLabel = 'X'     &
               ,yLabel = 'T',zLabel = 'U'
!               To take AV name of array
        Call faglStartWatch(fun, status)
!    Call StartWatch to let the axis-point know we're
!    interested in viewing fun
        Call faglStartWatch(X, status)
        Call faglStartWatch(T, status)
!
        Write(*,*) " Starting ARRAY Viewer "
!               Start AV by fav-programm
        Call favStartViewer(hv,status)
        If(status /= 0) Then
          Call favGetErrorNo(hv,nError,status)
               If(nError /= 0) Then
           Write(*,*) ' Array Viewer reports error = ',nError
               Stop
               End If
        End If
!               To take AV data
        Call favSetArray(hv,fun,status)
!               To take file name
        Call favSetArrayName(hv,"Line-1.agl",status)
!               To view array as vector graph
        Call favSetGraphType(hv,HeightPlot,status)
!               To set user's procedure named of axis
        Call favSetUseAxisLabel(hv,x_axis,1,status)
        Call favSetUseAxisLabel(hv,y_axis,1,status)
        Call favSetUseAxisLabel(hv,z_axis,1,status)
!               New (instead of "dim1") name x-axis.
        Call favSetAxisLabel(hv,x_axis,xLabel,status)
!               New (instead of "dim2") name y-axis.
        Call favSetAxisLabel(hv,y_axis,yLabel,status)
!               New (instead of "z-axis")       name z-axis.
        Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!               Setup the axis scales.
        Call favSetDimScale(hv, 2, X, status)
    Call favSetDimScale(hv, 1, T, status)
!
!               To display AV at screen
        Call favShowWindow(hv,av_true,status)
        Write(*,*) " Press any key to close down the viewer"
        key = getcharqq()
!               To close AV
        Call favEndViewer(hv,status)
        Call faglEndWatch(fun,status)
        Call faglEndWatch(X, status)
        Call faglEndWatch(T, status)
        End Subroutine v3D
```

2. This task shows simple programming of the standard procedure MOLCH in case when in (K-7.1) to the right the second order derivative is function. The function $u=u(x,t_{end})$ (Fig. K-7.2) is plotted with AV in the linear graph option (subroutine vGraph).

   Listing of computer code (FORTRAN-90) with user-friendly comments is given below.

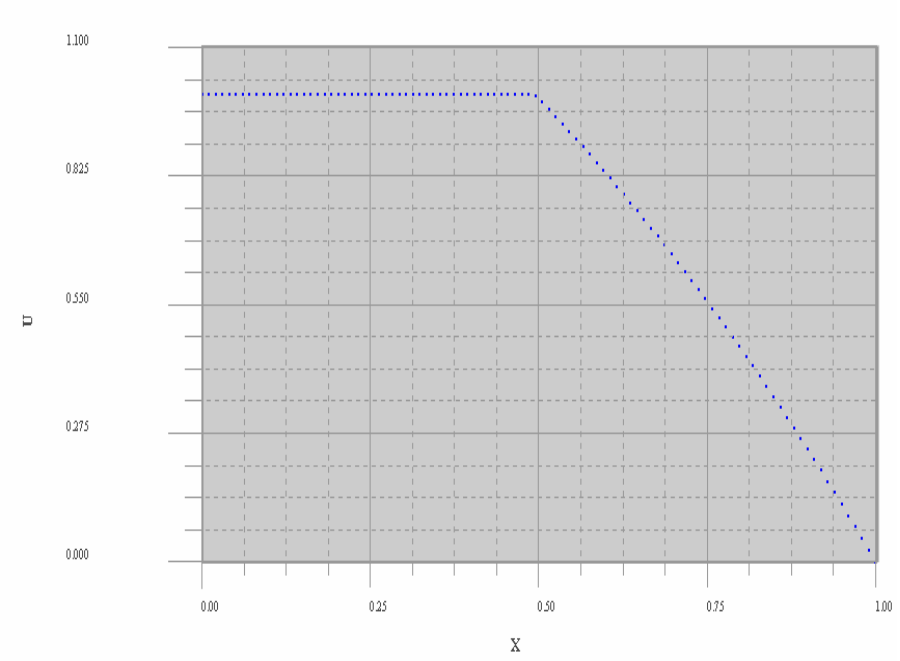


Fig. K-7.2. Function $u=u(x,t_{end})$ plotted by Array Visualization (AV).

```
        Program Line
!
!   The equation of diffusion-convection type with discontinuous
!   coefficients is solved (Problem C from Sincovec and Madsen, 1975).
!   This problem illustrates simple method for programming the
!   evaluation routine for the derivative Ut.
!
        Use DFIMSL
!
        Implicit none
!
!               SPECIFICATIONS FOR LOCAL VARIABLES
        INTEGER,PARAMETER :: NPDES = 1, NX = 100,          &
                             LDY = NPDES, NSTEP = 10
!               SPECIFICATIONS FOR LOCAL VARIABLES
        INTEGER I, IDO, J
        REAL U0, HINIT, T, TEND, TOL, XBREAK(NX), Y(LDY,NX)
!               Array for Array Viewer
        Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!               SPECIFICATIONS FOR FUNCTIONS
        EXTERNAL FCNBC, FCNUT
!
!       Open output file 'Line_21.rez'
        Open(2,file = 'Line_21.rez')
!
!               Set breakpoints and initial conditions
```

```
        U0 = 1.0
        Do I=1, NX
                XBREAK(I) = FLOAT(I-1)/(NX-1)
                Y(1,I) = 0.
        End Do
        Y(1,1) = U0
!
!               Set parameters for MOLCH
          TOL = SQRT(AMACH(4))
        HINIT = 0.01*TOL
            T = 0.0
          IDO = 1
!
        Do J = 1,NSTEP
!  Start Main cycle
!
        TEND = FLOAT(J)/FLOAT(NSTEP)
!
!               Solve the problem
        CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, &
                    XBREAK, TOL, HINIT, Y, LDY)
!               Print results
        If (J .EQ. 1) Then
        Write(2,'(/a)') '#    T          Y '
        Write(2,'(F9.2,100F9.2)') XBREAK(1),(XBREAK(i),i=1,Nx)
        End If
!    IF (J .EQ. NSTEP)                    &
                Write(2,'(F9.2,100F9.4)') T,(Y(1,i),i=1,Nx)
!
! Final call to release workspace
        IF (J .EQ. NSTEP) IDO = 3
        End Do
! End Main cycle
!
!       To use array visualizer
        allocate(UXTEND(2,NX))
        UXTEND(1,:) = XBREAK(1:NX)
        UXTEND(2,:) = Y(1,1:NX)
        Call vGraph(UXTEND,NX)
        deallocate(UXTEND)
!
        Close(2)
        Stop
        End Program Line
        SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!
        Implicit none
!
!               SPECIFICATIONS FOR ARGUMENTS
        INTEGER NPDES
        REAL X, T, D, V, U(*), UX(*), UXX(*), UT(*)
!
!               Define the PDE
! This is the nonlinear diffusion-convection with
! discontinuous coefficients.
!
        IF (X .LE. 0.5) THEN
                        D = 5.0 ; V = 1000.0
                                    ELSE
                        D = 1.0 ; V = 1.0
        END IF
        UT(1) = D*UXX(1) - V*UX(1)
        RETURN
        END SUBROUTINE FCNUT
```

```
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!
      Implicit none
!
!             SPECIFICATIONS FOR ARGUMENTS
      INTEGER NPDES
      REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!
!             SPECIFICATIONS FOR PARAMETERS
      ALPHA(1) = 1.0
       BETA(1) = 0.0
       GAMP(1) = 0.0
      RETURN
      END SUBROUTINE FCNBC

      Subroutine vGraph(fun,nValues)
!
!             To output array as vector graph
      Use avdef
      Use avviewer
      Use dflib
      Integer :: nValues
      Real    :: fun(2,nValues)
      Integer :: hv,status,nError
      Character(1) :: key
    Character(av_max_label_len) :: xLabel = 'X',yLabel = 'U'
!             To take AV name of array
      Call faglStartWatch(fun,status)
      Write(*,*) " Starting ARRAY Viewer "
!             Start AV by fav-programm
      Call favStartViewer(hv,status)
      If(status /= 0) Then
!
      Call favGetErrorNo(hv,nError,status)
             If(nError /= 0) Then
        Write(*,*) ' Array Viewer reports error = ',nError
             Stop
             End If
      End If
!
!             To take AV data
      Call favSetArray(hv,fun,status)
!             To take file name
      Call favSetArrayName(hv,"Line-2.agl",status)
!             To view array as vector graph
      Call favSetGraphType(hv,VectorGraph,status)
!             To set user's procedure named of axis
      Call favSetUseAxisLabel(hv,x_axis,1,status)
      Call favSetUseAxisLabel(hv,y_axis,1,status)
!             New (instead of "dim1") name x-axis.
      Call favSetAxisLabel(hv,x_axis,xLabel,status)
!             New (instead of "dim2") name y-axis.
      Call favSetAxisLabel(hv,y_axis,yLabel,status)
!             To display AV at screen
      Call favShowWindow(hv,av_true,status)
      Write(*,*) " Press any key to close down the viewer"
      key = getcharqq()

!         Close AV

      Call favEndViewer(hv,status)
      Call faglEndWatch(fun,status)
      End Subroutine vGraph
```

3. In this task additionally to the task 1 the initial data u_x are stated. It is done due to inaccuracy caused by derivative spline interpolation of the initial data stated in the form

$$u(0,t)=1+\cos[(2n-1)\pi x],\quad n>1,$$

to increase their accuracy. This initial data are compatible with the boundary conditions because the derivative

$$u_x(x,0)=-(2n-1)\pi\sin[(2n-1)\pi x] \tag{K-7.4}$$

becomes zero by $x=0$ and $x=1$.

Function $u=u(x,t)$ plotted with AV is given in Fig. K-7.3. The listing of computer code is given below.

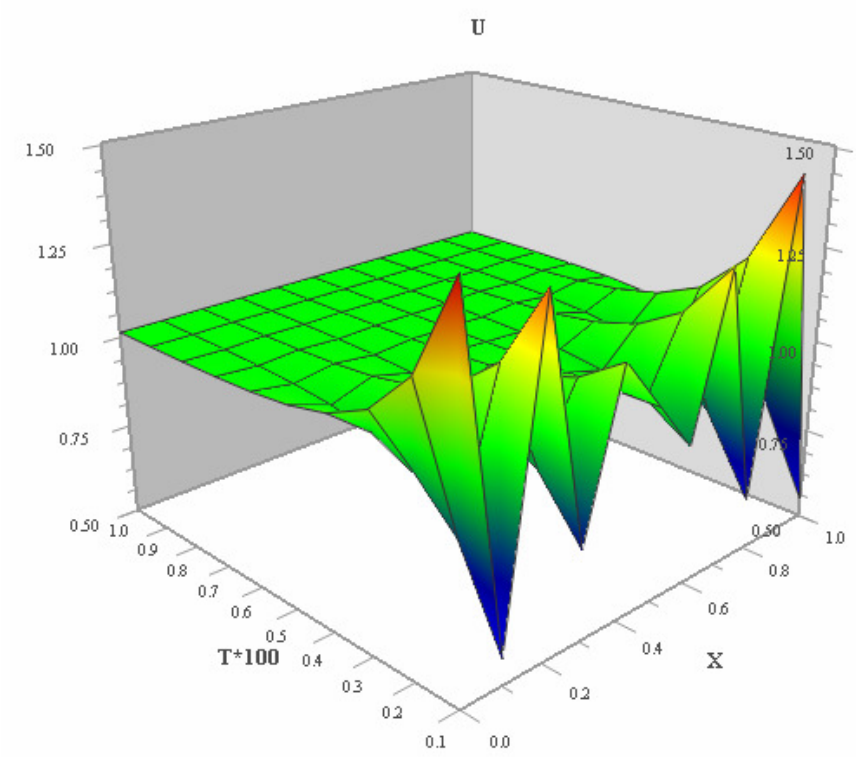


Fig. K-7.3. Function $u=u(x,t)$ plotted by AV.

```
Program Line
!
!  Linear normalized diffusion PDE Ut=Uxx is solved with MOLCH, but with
!  optional usage that provides values of the derivatives, Ux, of the
!  initial data. Due to errors in the numerical derivatives computed
!  by spline interpolation, more precise derivative values are required
```

```
!  when the initial data is U(x,0)=1+cos[(2n-1)pi*x], n > 1.
!  The boundary conditions are "zero flux": Ux(0,t)=Ux(1,t)=0, t>0.
!  Note: initial data is compatible with these ends conditions since the
!  derivative Ux(x,0)=-(2n-1)pi*Sin[(2n-1)pi*x] vanishes at x=0 and x=1.
!
   Use DFIMSL
   Implicit none
!
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER,PARAMETER :: NPDES=1, NX=11, LDY=NPDES, &
                        Nn = 5, Nstep = 10
!         SPECIFICATIONS FOR PARAMETERS
   INTEGER,PARAMETER :: ICHAP=5, Iget=1, Iput=2, &
                       kPARAM=11
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER I, ii, IDO, IOPT(1), J
   REAL ARG, HINIT, PARAM(50), PI, T, TEND, TOL, &
     &      XBREAK(NX),TSTEP(Nstep), Y(LDY,2*NX)
!
!  If you want to start array visualizer (AV), put AV_true=.True.
   Logical :: AV_true = .True.
!         Array for Array Viewer
   Real, Allocatable :: UXTEND(:,:)
!  dec$attributes array_visualizer :: UXTEND
!
!         SPECIFICATIONS FOR FUNCTIONS
   EXTERNAL FCNBC, FCNUT
!
!   Open output file 'Line_31.rez' and 'Line_31.dr'
   Open(2,file = 'Line_31.rez')
   Open(3,file = 'Line_31.dr')
!
!         Set breakpoints and initial conditions.
          PI = const('pi')
   IOPT(1) = kPARAM
!
   allocate(UXTEND(Nstep,NX))
!
   Do I=1, NX
          XBREAK(I) = FLOAT(I-1)/(NX-1)
                    ARG = (2.*Nn-1)*PI
!         Set function values.
             Y(1,I) = 1. + COS(ARG*XBREAK(I))
!         Set first derivative values.
          Y(1,I+NX) = - ARG*SIN(ARG*XBREAK(I))
   End Do
!         Set parameters for MOLCH
     TOL = SQRT(AMACH(4))
   HINIT = 0.01*TOL
          T = 0.0
     IDO = 1
! Get-reset PARAM to put the user-provided derivative of initial data.
   Call change_opt(Iget)
!   This flag signals that derivatives are passed.
   PARAM(17) = 1.
   Call change_opt(Iput)
! Look at output at steps of 0.001.
   TEND = 0.
   Do j = 1,Nstep
! Start main cycle
   TEND = TEND + 0.001
!         Solve the problem
   CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX,   &
    &                    XBREAK, TOL, HINIT, Y, LDY)
```

```
   UXTEND(j,:) = Y(1,1:NX)
      TSTEP(j) = T*100.
   If (J .EQ. 1) Then
   Write(2,'(/a)') "#    T        Y"
   Write(2,'(9x,11F9.4)') (XBREAK(i),i=1,Nx)
   Write(3,'(/a)') "#    T        Y'"
   Write(3,'(9x,11F9.4)') (XBREAK(i),i=1,Nx)
                End If
    Write(2,'(F9.3,11F9.4)') T,(Y(1,i),i=1,Nx)
    Write(3,'(F9.3,11F9.4)') T,(Y(1,i),ii=Nx+1,2*Nx)
!         Final call to release workspace
   IF (J .EQ. NSTEP) IDO = 3
!
   End     Do
!   End main cycle
!
!  Show, for example, the maximum step size used.
!
   Call change_opt(Iget)
   WRITE (2,*) '#  Maximum step size used is: ', &
               PARAM(33)
!         Reset option to defaults
   IOPT(1) = -IOPT(1)
   Call change_opt(Iput)
!
!  To use array visualizer
   If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
   Close(2)

   Contains

   Subroutine change_opt(iact)
! Internal routine to work options
     Integer iact
     CALL SUMAG ('math', ICHAP, iact, 1, IOPT, PARAM)
   End Subroutine change_opt

   End Program Line

   SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!         Define the PDE
   UT(1) = UXX(1)
   RETURN
   END SUBROUTINE FCNUT
!
   SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, ALPHA(*), BETA(*), GAMP(*)
   ALPHA(1) = 0.0
    BETA(1) = 1.0
    GAMP(1) = 0.0
   RETURN
   END SUBROUTINE FCNBC
   Subroutine v3D(fun,X,T,mValues,nValues)
!         To output array as 3D-figure
   Use avdef
   Use avviewer
   Use dflib
```

```
      Integer :: nValues,mValues
      Real    :: fun(mValues,nValues),X(nValues),T(mValues)
      Integer :: hv,status,nError
      Character(1) :: key
      Character(av_max_label_len) :: xLabel = 'X',yLabel = &
                                        'T*100',zLabel = 'U'
!           To take AV name of array
      Call faglStartWatch(fun, status)
!
!     StartWatch let the axis-point know we're interested in viewing fun
      Call faglStartWatch(X, status)
      Call faglStartWatch(T, status)
!
      Write(*,*) " Starting ARRAY Viewer "
!           Start AV by fav-programm
      Call favStartViewer(hv,status)
      If(status /= 0) Then
        Call favGetErrorNo(hv,nError,status)
           If(nError /= 0) Then
      Write(*,*) ' Array Viewer reports error = ',nError
           Stop
           End If
      End If
!           To take AV data
      Call favSetArray(hv,fun,status)
!           To take file name
      Call favSetArrayName(hv,"Line-3.agl",status)
!           To view array as vector graph
      Call favSetGraphType(hv,HeightPlot,status)
!           To set user's procedure named of axis
      Call favSetUseAxisLabel(hv,x_axis,1,status)
      Call favSetUseAxisLabel(hv,y_axis,1,status)
      Call favSetUseAxisLabel(hv,z_axis,1,status)
!           New (instead of "dim1")       name x-axis.
      Call favSetAxisLabel(hv,x_axis,xLabel,status)
!           New (instead of "dim2")       name y-axis.
      Call favSetAxisLabel(hv,y_axis,yLabel,status)
!           New (instead of "z-axis")     name z-axis.
      Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!           Setup the axis scales.
      Call favSetDimScale(hv, 2, X, status)
       Call favSetDimScale(hv, 1, T, status)
!
!           To display AV at screen
      Call favShowWindow(hv,av_true,status)
      Write(*,*) " Press any key to close down the viewer"
      key = getcharqq()
!           To close AV
      Call favEndViewer(hv,status)
      Call faglEndWatch(fun,status)
      Call faglEndWatch(X, status)
       Call faglEndWatch(T, status)
      End Subroutine v3D
```

4. Hyperbolic PDE is reduced to ordinary first order differential equation array introducing $u_t = v$. Then the second PDE of the system $v_t = u_{xx}$ is obtained. The initial data are $u(x,0)=Sin(\pi x)$ and $u_t(x,0)=v(x,0)=0$. The string ends are fixed. Thus, $u(0,t)=u(1,t)=v(0,t)=v(1,t)=0$. The known exact analytical solution

$$u(x,t) = Sin(\pi x)\, Cos(\pi x). \qquad \text{(K-7.5)}$$

allows calculating the Residuals at the output values of the numerical solution. Listing of FORTRAN-90 computer program with user-friendly comments is given below.

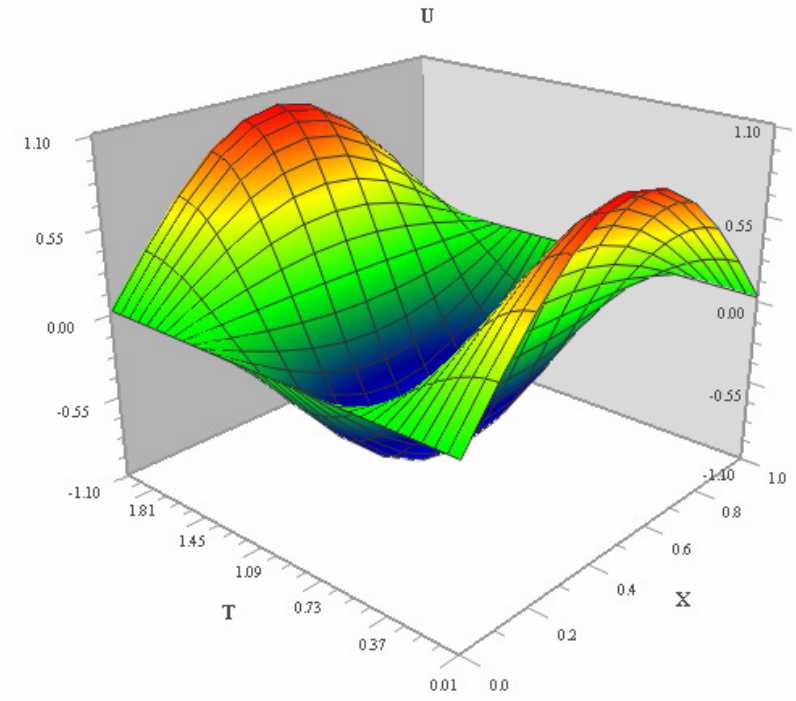


Fig.K-7.4. Function $u=u(x,t)$ plotted with AV.

```
   Program Line
!  Linear normalized hyperbolic PDE for the "vibrating string" Utt=Uxx
!  is solved. PDE is reduced to a system of first order ODEs. Define a
!  new dependent variable Ut=V. Then, Vt=Uxx is the second equation in
!  the system. Initial data U(x,0)=sin(pi*x) and Ut(x,0)=V(x,0)=0.
!  The ends of string are fixed: U(0,t)=U(1,t)=0 and V(0,t)=V(1,t)=0.
!  The exact solution is U(x,t)=sin(pi*x)*cos(pi*t).
!  Residuals are computed at the output values for 0 < t <= 2.
!  Output is obtained at 200 steps in increments of 0.01.
!  Though the sample code MOLCH gives satisfactory results for this
!  PDE, users should be aware that for non-linear problems, "shocks"
!  can appear in the solution, which may cause the code to fail in
!  unpredictable way. See Courant and Hilbert (1962), pp.488-490, for
!  an introductory discussion on shocks in hyperbolic systems.
!
   Use DFIMSL
!
   Implicit none
!
!         SPECIFICATIONS FOR LOCAL VARIABLES
    INTEGER,PARAMETER :: NPDES=2, NX=11, LDY=NPDES, Nstep = 200
```

```
!          SPECIFICATIONS FOR PARAMETERS
   INTEGER,PARAMETER :: ICHAP=5, IGET=1, IPUT=2, kPARAM=11
!          SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER I, IDO, IOPT(1), J, ii
   REAL HINIT, PARAM(50), Pi, T, TEND, TOL, XBREAK(NX), &
   &        Y(LDY,2*NX), ERROR(NX), TSTEP(Nstep), ERRU
!
!  To start array visualizer (AV), put AV_true = .True.
   Logical :: AV_true = .True.
!          Array for Array Viewer
   Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!          SPECIFICATIONS FOR FUNCTIONS
   EXTERNAL FCNBC, FCNUT
!
!   Open output file 'Line_41.rez' and 'Line_41.dr'
   Open(2,file = 'Line_41.rez')
   Open(3,file = 'Line_41.dr')
!
!          Set breakpoints and initial conditions.
!
   Pi = CONST('pi')
   IOPT(1) = kPARAM
!
   allocate(UXTEND(Nstep,NX))
!
   Do I=1, NX
   XBREAK(I) = FLOAT(I-1)/(NX-1)
!          Set function values.
   Y(1,I) = SIN(Pi*XBREAK(I))
   Y(2,I) = 0.
!          Set first derivative values.
   Y(1,I+NX) = Pi*COS(Pi*XBREAK(I))
   Y(2,I+NX) = 0.0
   End Do
!          Set parameters for MOLCH
     TOL = 0.1*SQRT(AMACH(4))
   HINIT = 0.01*TOL
       T = 0.0
     IDO = 1
! Get and reset the PARAM array so that user-provided
!          derivatives of the initial data are used.
   Call change_opt(Iget)
!          This flag signals that derivatives are passed.
   PARAM(17) = 1.
   Call change_opt(Iput)
! Look at output at steps of 0.001 and compute errors.
   ERRU = 0. ; TEND = 0.
!
   Do j = 1,Nstep         ! Start main cycle
!
   TEND = TEND + 0.01
!          Solve the problem
   CALL MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, NX, &
   &           XBREAK, TOL, HINIT, Y, LDY)
   UXTEND(j,:) = Y(1,1:NX)
      TSTEP(j) = T !*100.
   If (J .EQ. 1) Then
   Write(2,'(/a)') "#    T         Y"
   Write(2,'(9x,11F9.4)') (XBREAK(i),i=1,Nx)
   Write(3,'(/a)') "#    T         Y'"
   Write(3,'(9x,11F9.4)') (XBREAK(i),i=1,Nx)
                End If
```

```
    Write(2,'(F9.3,11F9.4)') T,(Y(1,i),i=1,Nx)
    Write(3,'(F9.3,11F9.4)') T,(Y(1,i),ii=Nx+1,2*Nx)
!
   Do I=1, NX
   ERROR(I) = Y(1,I) - SIN(Pi*XBREAK(I))*COS(Pi*TEND)
   End Do
   Do I=1, NX
          ERRU = AMAX1(ERRU,ABS(ERROR(I)))
   End Do
!         Final call to release workspace
   IF (J .EQ. NSTEP) IDO = 3
!
   End    Do
!   End main cycle
!
!         Show, for example, the maximum step size used.
!
   Call change_opt(Iget)
   WRITE (2,*) '#  Maximum error in U(x,t) ', &
   & 'divided by TOL: ',ERRU/TOL
    WRITE (2,*) '#  Maximum step size used is: ', PARAM(33)
!         Reset option to defaults
   IOPT(1) = -IOPT(1)
   Call change_opt(Iput)
!
!         To use array visualizer
   If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
!
   Close(2)
   STOP

   Contains

   Subroutine change_opt(iact)
!         Internal routine to work options
     Integer iact
     CALL SUMAG ('math', ICHAP, iact, 1, IOPT, PARAM)
   End Subroutine change_opt

   End Program Line
!
   SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!
   Implicit none
!
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!         Define the PDE
   UT(1) = U(2)
   UT(2) = UXX(1)
   RETURN
   END SUBROUTINE FCNUT

   SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!
   Implicit none
!
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, ALPHA(*), BETA(*), GAMP(*)
   ALPHA(1) = 1.0
```

```
      BETA(1) = 0.0
      GAMP(1) = 0.0
      ALPHA(2) = 1.0
      BETA(2) = 0.0
      GAMP(2) = 0.0
      RETURN
      END SUBROUTINE FCNBC
!
      Subroutine v3D(fun,X,T,mValues,nValues)
!         To output array as 3D-figure
      Use avdef
      Use avviewer
      Use dflib
      Integer :: nValues,mValues
      Real    :: fun(mValues,nValues),X(nValues),T(mValues)
      Integer :: hv,status,nError
      Character(1) :: key
      Character(av_max_label_len) :: xLabel = 'X'  ?
                        ,yLabel = 'T',zLabel = 'U'
!         To take AV name of array
      Call faglStartWatch(fun, status)
!
!   StartWatch lets the axis-point know we're interested in viewing fun
      Call faglStartWatch(X, status)
      Call faglStartWatch(T, status)
!
      Write(*,*) " Starting ARRAY Viewer "
!         Start AV by fav-programm
      Call favStartViewer(hv,status)
      If(status /= 0) Then
        Call favGetErrorNo(hv,nError,status)
         If(nError /= 0) Then
      Write(*,*) ' Array Viewer reports error = ',nError
         Stop
         End If
      End If
!         To take AV data
      Call favSetArray(hv,fun,status)
!         To take file name
      Call favSetArrayName(hv,"Line-4.agl",status)
!         To view array as vector graph
      Call favSetGraphType(hv,HeightPlot,status)
!         To set user's procedure named of axis
      Call favSetUseAxisLabel(hv,x_axis,1,status)
      Call favSetUseAxisLabel(hv,y_axis,1,status)
      Call favSetUseAxisLabel(hv,z_axis,1,status)
!         New (instead of "dim1")       name x-axis.
      Call favSetAxisLabel(hv,x_axis,xLabel,status)
!         New (instead of "dim2")       name y-axis.
      Call favSetAxisLabel(hv,y_axis,yLabel,status)
!         New (instead of "z-axis")     name z-axis.
      Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!         Setup the axis scales.
      Call favSetDimScale(hv, 2, X, status)
       Call favSetDimScale(hv, 1, T, status)
!
!         To display AV at screen
      Call favShowWindow(hv,av_true,status)
      Write(*,*) " Press any key to close down the viewer"
      key = getcharqq()
!         To close AV
      Call favEndViewer(hv,status)
      Call faglEndWatch(fun,status)
```

```
   Call faglEndWatch(X, status)
       Call faglEndWatch(T, status)
   End Subroutine v3D
```

5. First the equation (4.1) is normalized using $t_{max} = 1/\alpha$. The numerical solution is compared against the exact analytical solution (K-4.3) obtained by separated variables. The mean-square error is computed with (K-4.4). RMS=5.4050D-03 is here that is less than mean-square error for the most of considered above finite-difference schemes being independent of time-step.

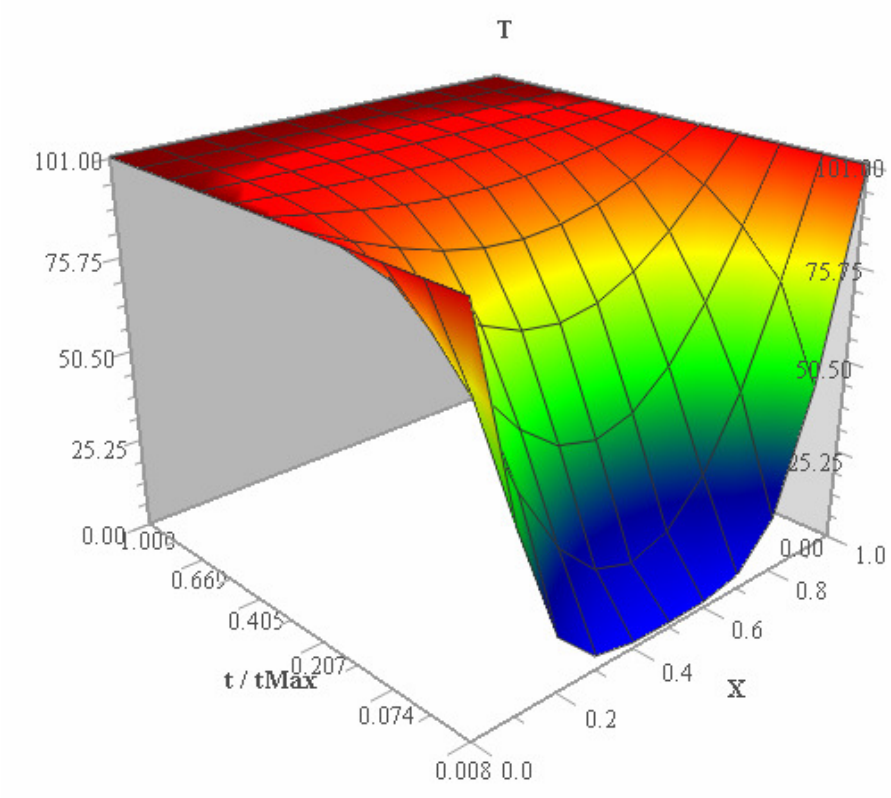


Fig. K-7.5. Solution $T=T(x,t)$ plotted with AV.

Listing of computer code in FORTRAN-90 with comments is given below.

```
   Program Line
!
   Use DFIMSL
!
   Implicit none
!  SPECIFICATIONS FOR LOCAL VARIABLES
```

```
      INTEGER,PARAMETER :: NPDES = 1, Nx = 11, LDY = NPDES
!     MaxEx- max number of terms in the exact solution
      Integer,Parameter :: NSTEP = 11, MaxEx = 10
!           SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER I, IDO, J
!
      Real,Parameter :: tMax = 6000.0 ! Max Time
!     Alph - the thermal diffusivity
      Real,Parameter :: Alph = 1./6000.0 ! = 1./tMax
      Real,Parameter :: Co   = 100.0  ! grad C
      Real,Parameter :: aL   = 1.0    ! wall thickness
!
      REAL HINIT, T, TEND, TOL, U0, Pi, ajm
      Real TSTEP(NSTEP), TE(Nx), TD(Nx), XBREAK(Nx), ?
               Y(LDY,Nx)
      Real*8 sum,avs,rms,dmp
!     If you want to start array visualizer (AV) to put
1     AV_true = .True.
      Logical :: AV_true = .True.
!           Array for Array Viewer
      Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!           SPECIFICATIONS FOR FUNCTIONS
      EXTERNAL FCNBC, FCNUT
!
!      Open output file 'Line_71.rez'
      Open(2,file = 'Line_71.rez')
!
!           Set breakpoints and initial conditions
       U0 = 0.0
       Pi = const('pi')
      ajm = Nstep - 1
      Do I=1, Nx
           XBREAK(I) = FLOAT(I-1)/(NX-1)
           Y(1,I) = U0
      End Do
      Y(1,1) = 1.0 ; Y(1,Nx) = 1.0
!           Set parameters for MOLCH
      TOL = SQRT(AMACH(4))
      HINIT = 0.01*TOL
      T = 0.0
      IDO = 1
!
      allocate(UXTEND(NSTEP,NX))
!
      Do j= 1,NSTEP  ! Start Main cycle
!
      TEND = FLOAT(J)/FLOAT(NSTEP)
!
!          This puts more output for small t values
!          where action is fastest.
!
      TEND = TEND*TEND
!
!           Solve the problem
      Call MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, Nx,   &
        &                  XBREAK, TOL,HINIT, Y, LDY)
      UXTEND(j,:) = Co*Y(1,1:Nx)
         TD(1:Nx) = Co*Y(1,1:Nx)
         TSTEP(j) = T
!           Print results
      If (J .EQ. 1) Then
      Write(2,'(/a)') '#    T         Y '
```

```
   Write(2,'(9x,11F9.2)') (XBREAK(i),i=1,Nx)
                End If
    Write(2,'(F9.2,11F9.4)') T,(Y(1,i),i=1,Nx)
!
   If(j == Nstep) Then
!
   Call EXACT
   do i = 1,Nstep
          dmp = TE(i) - TD(i)
          sum = sum + dmp*dmp
   end do
   Write(2,100) tMax*t,(TD(i),i=1,Nx)
100Format(/,' T= ',F7.2,' TD= ',11F7.2)
   Write(2,101) tMax*t,(TE(i),i=1,Nx)
101Format(/,' T= ',F7.2,' TE= ',11F7.2,//)
!
!  rms is the RMS error
!
   avs = sum/(1.+ ajm)
   rms = dsqrt(avs)
   Write(2,'(/a,1pD11.4,/)') ' RMS Line = ',rms
!          Final call to release workspace
   IDO = 3
   End If

   End Do
! End Main cycle
!
!  To use array visualizer
   If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
!
   Close(2)

   Stop

   Contains

   Subroutine EXACT
!
!  Exact solution of the transient heat conduction problem
!
   Integer i,m
   Real dam,am,dxm,dtm,aj

   TE = 100.0
   Do i = 1,Nx
          Do m = 1,MaxEx
            am = m
           dam = 2.*am - 1
          dxm = dam*Pi*XBREAK(i)
          dtm = - Alph*dam*dam*Pi*Pi*tMax*t
!
!  Limit the argument size of exp(dtm)
!
          If(dtm .LT. - 87.0) dtm = - 87.0
                 dtm = exp(dtm)
          If(dtm .LE. 1.0e-10) cycle
     TE(i) = TE(i) - 400./dam/Pi*sin(dxm)*dtm
       End Do
   End Do
   End Subroutine EXACT
   End Program Line
   SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
```

```
      Implicit none
!
      Real,Parameter :: Alph = 1.0
!           SPECIFICATIONS FOR ARGUMENTS
      INTEGER NPDES
      REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!           Define the PDE
      UT(1) = Alph*UXX(1)
      RETURN
      END SUBROUTINE FCNUT
!
      SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!           SPECIFICATIONS FOR ARGUMENTS
      INTEGER NPDES
      REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!
      ALPHA(1) = 1.0
       BETA(1) = 0.0
       GAMP(1) = 0.0
      RETURN
      END SUBROUTINE FCNBC

      Subroutine v3D(fun,X,T,mValues,nValues)
!           To output array as 3D-figure
      Use avdef
      Use avviewer
      Use dflib
      Integer :: nValues,mValues
      Real    :: fun(mValues,nValues),X(nValues),T(mValues)
      Integer :: hv,status,nError
      Character(1) :: key
      Character(av_max_label_len) :: xLabel = 'X', ?
                  yLabel = 't / tMax',zLabel = 'T'
!           To take AV name of array
      Call faglStartWatch(fun, status)
!
!  Call StartWatch to let the axis-point know we're
!     interested in viewing fun
      Call faglStartWatch(X, status)
      Call faglStartWatch(T, status)
!
      Write(*,*) " Starting ARRAY Viewer "
!           Start AV by fav-programm
      Call favStartViewer(hv,status)
      If(status /= 0) Then
        Call favGetErrorNo(hv,nError,status)
           If(nError /= 0) Then
      Write(*,*) ' Array Viewer reports error = ',nError
           Stop
           End If
      End If
!           To take AV data
      Call favSetArray(hv,fun,status)
!           To take file name
      Call favSetArrayName(hv,"Line-7.agl",status)
!           To view array as vector graph
      Call favSetGraphType(hv,HeightPlot,status)
!           To set user's procedure named of axis
      Call favSetUseAxisLabel(hv,x_axis,1,status)
      Call favSetUseAxisLabel(hv,y_axis,1,status)
      Call favSetUseAxisLabel(hv,z_axis,1,status)
!           New (instead of "dim1")       name x-axis.
      Call favSetAxisLabel(hv,x_axis,xLabel,status)
```

```
!          New (instead of "dim2")       name y-axis.
   Call favSetAxisLabel(hv,y_axis,yLabel,status)
!          New (instead of "z-axis")     name z-axis.
   Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!          Setup the axis scales.
   Call favSetDimScale(hv, 2, X, status)
         Call favSetDimScale(hv, 1, T, status)
!
!          To display AV at screen
   Call favShowWindow(hv,av_true,status)
   Write(*,*) " Press any key to close down the viewer"
   key = getcharqq()
!          To close AV
   Call favEndViewer(hv,status)
   Call faglEndWatch(fun,status)
   Call faglEndWatch(X, status)
         Call faglEndWatch(T, status)
   End Subroutine v3D
```

6. The PDE (4.1) is normalized introducing $t_{max} = 1/\alpha$. The solution is compared to the exact analytical solution and the mean-square error is computed with (K-4.4). RMS=5.7120D-03 is here that is less than mean-square error for the most of considered above finite-difference schemes (see Tables from the tasks 5-9 in the chapter 4) being independent of time-step.

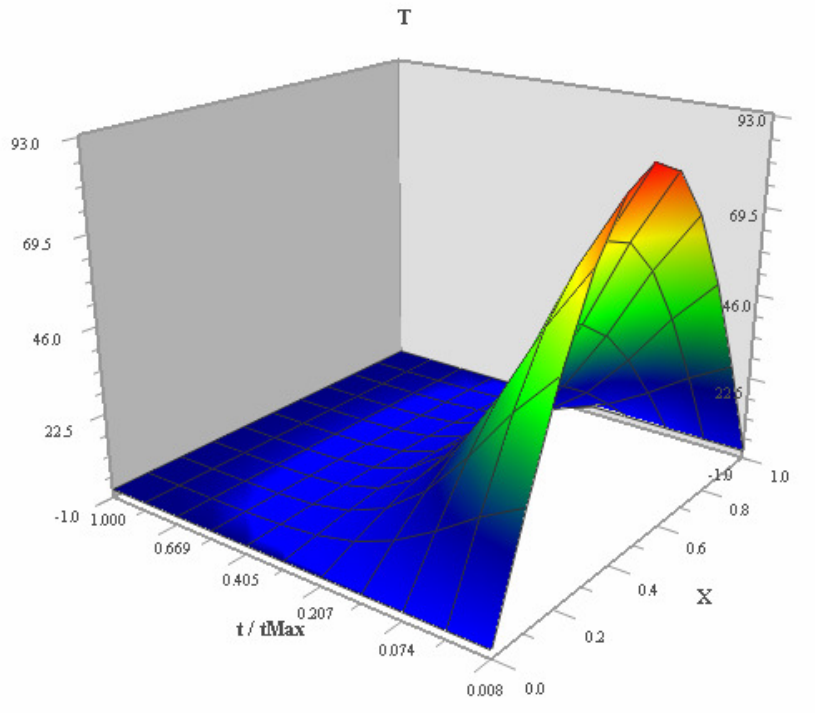


Fig. K-7.6. Solution $T=T(x,t)$ plotted with AV.

The listing of the computer code (FORTRAN-90) with comments is given below.

```
   Program Line
!
   Use DFIMSL
!
   Implicit none
!
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER,PARAMETER :: NPDES = 1, Nx = 11, LDY = NPDES
!  MaxEx- max number of terms in the exact solution
   Integer,Parameter :: NSTEP = 11, MaxEx = 10
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER I, IDO, J
   Real,Parameter :: tMax = 10.0 ! Max Time
!  Alph - the thermal diffusivity
   Real,Parameter :: Alph = 1./10.0 ! = 1./tMax
   Real,Parameter :: Co   = 100.0  ! grad C
   Real,Parameter :: aL   = 1.0    ! wall thickness
!
   REAL HINIT, T, TEND, TOL, U0, Pi, ajm
   Real TSTEP(NSTEP), TE(Nx), TD(Nx), XBREAK(Nx), ?
          Y(LDY,Nx)
   Real*8 sum,avs,rms,dmp
!  If you want to start array visualizer (AV) to put
!  AV_true = .True.
   Logical :: AV_true = .True.
!         Array for Array Viewer
   Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!         SPECIFICATIONS FOR FUNCTIONS
   EXTERNAL FCNBC, FCNUT
!
!   Open output file 'Line_81.rez'
   Open(2,file = 'Line_81.rez')
!
!         Set breakpoints and initial conditions
    U0 = 0.0
    Pi = const('pi')
   ajm = Nstep - 1
   Do I=1, Nx
          XBREAK(I) = FLOAT(I-1)/(NX-1)
          Y(1,I) = Sin(Pi*XBREAK(I)/aL)
   End Do
   Y(1,1) = 0.0 ; Y(1,Nx) = 0.0
!         Set parameters for MOLCH
   TOL = SQRT(AMACH(4))
   HINIT = 0.01*TOL
   T = 0.0
   IDO = 1
!
   allocate(UXTEND(NSTEP,NX))
!
   Do j= 1,NSTEP
! Start Main cycle
!
   TEND = FLOAT(J)/FLOAT(NSTEP)
!
! This puts more output for small t values where action is fastest.
!
   TEND = TEND*TEND
```

```
!         Solve the problem
   Call MOLCH (IDO, FCNUT, FCNBC, NPDES, T, TEND, Nx, ?
                    XBREAK, TOL, HINIT, Y, LDY)
!
   UXTEND(j,:) = Co*Y(1,1:Nx)
      TD(1:Nx) = Co*Y(1,1:Nx)
      TSTEP(j) = T
!         Print results
   If (J .EQ. 1) Then
   Write(2,'(/a)') '#    T          Y '
   Write(2,'(9x,11F9.2)') (XBREAK(i),i=1,Nx)
                End If
    Write(2,'(F9.2,11F9.4)') T,(Y(1,i),i=1,Nx)
!
   If(j == Nstep) Then
!
   Call EXACT
!
   do i = 1,Nstep
          dmp = TE(i) - TD(i)
          sum = sum + dmp*dmp
   end do
   Write(2,100) tMax*t,(TD(i),i=1,Nx)
100Format(/,' T= ',F7.2,' TD= ',11F7.2)
   Write(2,101) tMax*t,(TE(i),i=1,Nx)
101Format(/,' T= ',F7.2,' TE= ',11F7.2,//)
!
!  rms is the RMS error
!
   avs = sum/(1.+ ajm)
   rms = dsqrt(avs)
   Write(2,'(/a,1pD11.4,/)') ' RMS Line = ',rms
!         Final call to release workspace
   IDO = 3
   End If
   End Do
! End Main cycle
!
!  To use array visualizer
   If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
!
   Close(2)

   Stop

   Contains

   Subroutine EXACT
!
! Exact solution of the transient heat conduction problem
!
   Integer i,m
   Real dam,am,dxm,dtm,aj

   TE = 100.0
   Do i = 1,Nx
          dxm = Pi*XBREAK(i)/aL
          dtm = -Alph*Pi*Pi*t*tMax/aL/aL
!
!  Limit the argument size of exp(dtm)
!
          If(dtm .LT. - 75.0) dtm = - 75.0
                 dtm = exp(dtm)
```

```
            If(t .LE. 0.0) dtm = 1.0
      TE(i) = Co*dtm*sin(dxm)

   End Do
   End Subroutine EXACT
   End Program Line
!
   SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!
   Implicit none
!
   Real,Parameter :: Alph = 1.0
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!         Define the PDE
   UT(1) = Alph*UXX(1)
   RETURN
   END SUBROUTINE FCNUT
!
   SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!
   ALPHA(1) = 1.0
    BETA(1) = 0.0
    GAMP(1) = 0.0
   RETURN
   END SUBROUTINE FCNBC

   Subroutine v3D(fun,X,T,mValues,nValues)
!         To output array as 3D-figure
   Use avdef
   Use avviewer
   Use dflib
   Integer :: nValues,mValues
   Real    :: fun(mValues,nValues),X(nValues),T(mValues)
   Integer :: hv,status,nError
   Character(1) :: key
   Character(av_max_label_len) :: xLabel = 'X', &
               yLabel = 't / tMax', zLabel = 'T'
!         To take AV name of array
   Call faglStartWatch(fun, status)
!
! StartWatch to let the axis-point know we're interested in viewing fun
   Call faglStartWatch(X, status)
   Call faglStartWatch(T, status)
!
   Write(*,*) " Starting ARRAY Viewer "
!         Start AV by fav-programm
   Call favStartViewer(hv,status)
   If(status /= 0) Then
     Call favGetErrorNo(hv,nError,status)
           If(nError /= 0) Then
   Write(*,*) ' Array Viewer reports error = ',nError
           Stop
           End If
   End If
!         To take AV data
   Call favSetArray(hv,fun,status)
!         To take file name
   Call favSetArrayName(hv,"Line-8.agl",status)
```

```
!          To view array as vector graph
   Call favSetGraphType(hv,HeightPlot,status)
!          To set user's procedure named of axis
   Call favSetUseAxisLabel(hv,x_axis,1,status)
   Call favSetUseAxisLabel(hv,y_axis,1,status)
   Call favSetUseAxisLabel(hv,z_axis,1,status)
!          New (instead of "dim1")       name x-axis.
   Call favSetAxisLabel(hv,x_axis,xLabel,status)
!          New (instead of "dim2")       name y-axis.
   Call favSetAxisLabel(hv,y_axis,yLabel,status)
!          New (instead of "z-axis")     name z-axis.
   Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!          Setup the axis scales.
   Call favSetDimScale(hv, 2, X, status)
   Call favSetDimScale(hv, 1, T, status)
!
!          To display AV at screen
   Call favShowWindow(hv,av_true,status)
   Write(*,*) " Press any key to close down the viewer"
   key = getcharqq()
!          To close AV
   Call favEndViewer(hv,status)
   Call faglEndWatch(fun,status)
   Call faglEndWatch(X, status)
   Call faglEndWatch(T, status)
!
   End Subroutine v3D
```

7. PDE (4.1) is normalized introducing $t_{max} = 1/u$. The solution is compared to the exact analytical solution and the mean-square error is computed with (K-4.4).

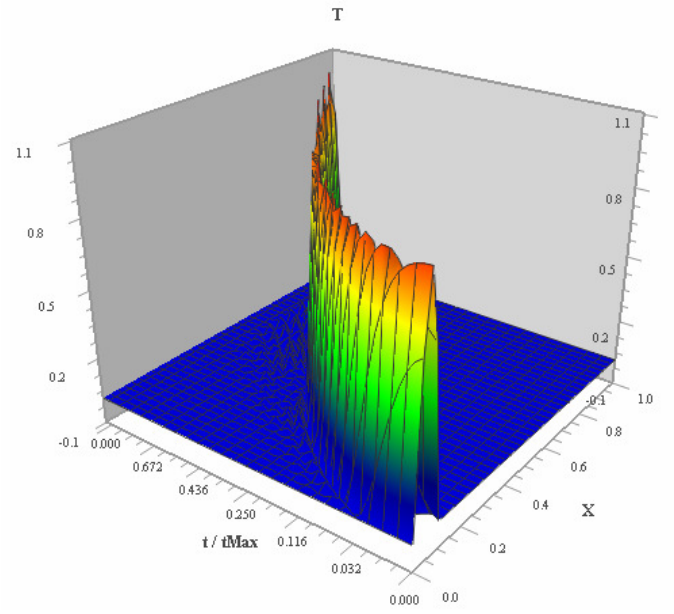


Fig. K-7.7. Numerical solution plotted with AV.

RMS=2.5099D-02 that is less than mean-square error for the most of considered above finite-difference schemes. The solution behaviour is clearly observed from the Fig. K-7.7, which shows much less oscillations than for the most of the above-considered numerical schemes.

The listing of the computer code (FORTRAN-90) with comments is given below.

```
   Program Line
!
   Use DFIMSL
!
   Implicit none
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER,PARAMETER :: nPDES = 1, Nx = 41, Ldy = nPDES
!  MaxEx- max number of terms in the exact solution
   Integer,Parameter :: Nstep = 50, Nst = 40
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER i, iDO, j
   Real,Parameter :: tMax = 10.0    ! Max Time
!  Alph - the thermal diffusivity
   Real,Parameter :: Alph = 1./10.0 ! = 1./tMax
   Real,Parameter :: Co   = 1.0     ! grad C
   Real,Parameter :: aL   = 1.0     ! wall thickness
   Real,Parameter ::  U   = 0.1     ! velosity
   REAL hINIT, t, Tend, tOL, Pi, ajm, aim, dX, x
     Real TSTEP(Nstep), TE(Nx), TD(Nx), XBREAK(Nx), Y(Ldy,Nx)
   Real*8 sum,avs,rms,dmp
!  If you want to start array visualizer (AV)
!  to put AV_true = .True.
   Logical :: AV_true = .True.
!         Array for Array Viewer
   Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!         SPECIFICATIONS FOR FUNCTIONS
   EXTERNAL FCNBC, FCNUT
!   Open output file 'Line_91.rez'
   Open(2,file = 'Line_91.rez')
!
!         Set breakpoints and initial conditions
    Pi = const('pi')
   ajm = Nstep - 1
   aim = Nx - 1
    dX = aL/aim
   Y(1,1:Nx) = 0.0
   Do i = 1,Nx
                  x = FLOAT(i-1)/aim
         XBREAK(i) = x
    If(x .LE. 0.1) Y(1,i) = Sin(10.*Pi*x)
   End Do
   Y(1,1) = 0.0 ; Y(1,Nx) = 0.0
! Set parameters for MOLCH
   tOL = SQRT(AMACH(4))
   hINIT = 0.01*tOL
   t = 0.0
   iDO = 1
   allocate(UXTEND(Nstep,Nx))
   UXTEND = 0.0
   Do j= 1,Nstep
! Start Main cycle
   Tend = FLOAT(j)/FLOAT(Nstep)
```

```
! This puts more output for small t values where action is fastest.
!
   Tend = Tend*Tend
!
! Solve the problem
   Call MOLCH (iDO, FCNUT, FCNBC, nPDES, t, Tend, Nx, &
     &                  XBREAK, tOL, hINIT, Y, Ldy)
!
   If(iDO == 3) Exit

   UXTEND(j,:) = Co*Y(1,1:Nx)
      TD(1:Nx) = Co*Y(1,1:Nx)
      TSTEP(j) = t
!         Print results
   If (J .EQ. 1) Then
   Write(2,'(/a)') '#    t         Y '
   Write(2,'(9x,11F9.2)') (XBREAK(i),i=1,Nx,4)
                End If
    Write(2,'(F9.2,11F9.4)') t,(Y(1,i),i=1,Nx,4)
!
   If(t >= 0.8) Then
   Call EXACT
!
   do i = 1,Nx
           dmp = TE(i) - TD(i)
           sum = sum + dmp*dmp
   end do
   Write(2,100) tMax*t,(TD(i),i=1,Nx,4)
100Format(/,' T= ',F7.2,' TD= ',11F7.2)
   Write(2,101) tMax*t,(TE(i),i=1,Nx,4)
101Format(/,' T= ',F7.2,' TE= ',11F7.2,//)
!
!  rms is the RMS error
   avs = sum/(1.+ ajm)
   rms = dsqrt(avs)
   Write(2,'(/a,1pD11.4,/)') ' RMS Line = ',rms
! Final call to release workspace
   iDO = 3
   End If
   End Do
! End Main cycle
!
!  To use array visualizer
   If(AV_true) Call v3D(UXTEND,XBREAK,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
!
   Close(2)
   Stop

   Contains

   Subroutine EXACT
! Exact solution of the transient heat conduction problem
!
   Integer i,ip,Jm,inc
   Real ai, dxm, tN, x0, u0
    TE = 0.0
    u0 = 1.0
   Do i = 1,Nx
            ai = i - 1
        x0 = dX*ai
            If(x0 > u0*t .AND. x0 <= u0*t+0.1) Then
           dxm = 10.0*Pi*(x0 - u0*t)
     TE(i) = sin(dxm)
```

```
          End If
        End Do
    End Subroutine EXACT
    End Program Line
!
    SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
    Implicit none
!   Real,Parameter :: Alph = 1.0
    Real,Parameter :: c = 1.0
!         SPECIFICATIONS FOR ARGUMENTS
    INTEGER NPDES
    REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!         Define the PDE
    UT(1) = - c*UX(1) !Alph*UXX(1)
    RETURN
    END SUBROUTINE FCNUT
!
    SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!         SPECIFICATIONS FOR ARGUMENTS
    INTEGER NPDES
    REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!
    ALPHA(1) = 1.0
     BETA(1) = 0.0
     GAMP(1) = 0.0
    RETURN
    END SUBROUTINE FCNBC

    Subroutine v3D(fun,X,T,mValues,nValues)
!         To output array as 3D-figure
    Use avdef
    Use avviewer
    Use dflib
    Integer :: nValues,mValues
    Real    :: fun(mValues,nValues),X(nValues),T(mValues)
    Integer :: hv,status,nError
    Character(1) :: key
    Character(av_max_label_len) :: xLabel = 'X', &
                        yLabel = 't / tMax', zLabel = 'T'
!         To take AV name of array
    Call faglStartWatch(fun, status)
!
! StartWatch to let the axis-point know we're interested in viewing fun
    Call faglStartWatch(X, status)
    Call faglStartWatch(T, status)
!
    Write(*,*) " Starting ARRAY Viewer "
!         Start AV by fav-programm
    Call favStartViewer(hv,status)
    If(status /= 0) Then
      Call favGetErrorNo(hv,nError,status)
         If(nError /= 0) Then
    Write(*,*) ' Array Viewer reports error = ',nError
         Stop
         End If
    End If
!         To take AV data
    Call favSetArray(hv,fun,status)
!         To take file name
    Call favSetArrayName(hv,"Line-9.agl",status)
!         To view array as vector graph
    Call favSetGraphType(hv,HeightPlot,status)
!         To set user's procedure named of axis
```

```
   Call favSetUseAxisLabel(hv,x_axis,1,status)
   Call favSetUseAxisLabel(hv,y_axis,1,status)
   Call favSetUseAxisLabel(hv,z_axis,1,status)
!         New (instead of "dim1")       name x-axis.
   Call favSetAxisLabel(hv,x_axis,xLabel,status)
!         New (instead of "dim2")       name y-axis.
   Call favSetAxisLabel(hv,y_axis,yLabel,status)
!         New (instead of "z-axis")     name z-axis.
   Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!         Setup the axis scales.
   Call favSetDimScale(hv, 2, X, status)
         Call favSetDimScale(hv, 1, T, status)
!
!         To display AV at screen
   Call favShowWindow(hv,av_true,status)
   Write(*,*) " Press any key to close down the viewer"
   key = getcharqq()
!         To close AV
   Call favEndViewer(hv,status)
   Call faglEndWatch(fun,status)
   Call faglEndWatch(X, status)
         Call faglEndWatch(T, status)
   End Subroutine v3D
```

8. First the equation (4.1) is normalized using the transformation $u = u'/L$, $\alpha = \alpha'/L$. Here L is the width of the wave front, $u'=0.5$ and $\alpha'=0.1$ are the wave speed and diffusion coefficient, respectively.

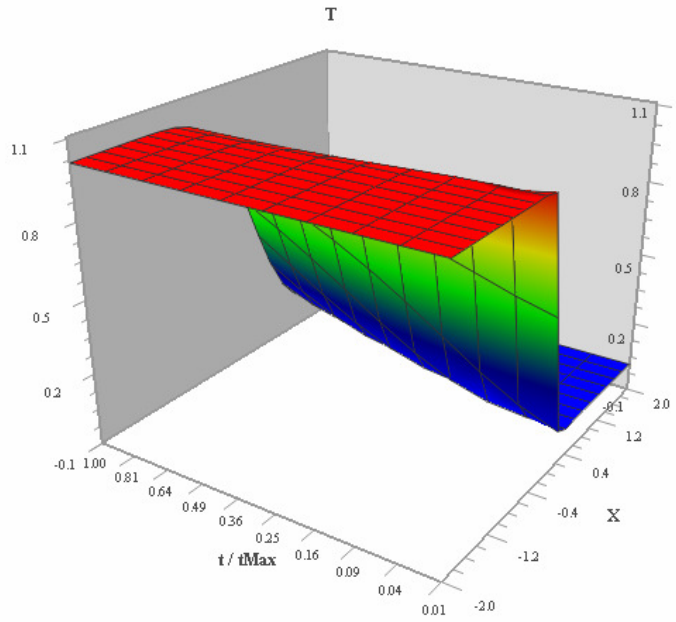


Fig. K-7.8. Numerical solution $T=T(x,t)$ plotted with Array Visualizer (AV).

The solution is compared to the exact analytical solution obtained with the method of separated variables. The mean-square error computed with (K-4.4) is RMS=3.0003D-03, which is less than mean-square error for the most of the considered above finite-difference schemes.

The numerical solution behaviour is clearly observed from the Fig. K-7.8. It is shown that the numerical oscillations for this scheme are much less than for the most of the above considered other numerical schemes.

The listing of the computer code (FORTRAN-90) with user-friendly comments is given below.

```
   Program Line
!
   Use DFIMSL
!
   Implicit none
!
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER,PARAMETER :: nPDES = 1, Nx = 21, Ldy = nPDES
!  MaxEx- max number of terms in the exact solution
   Integer,Parameter :: Nstep = 10
!  lN   - max number of member series of exact solution
   Integer,Parameter ::  lN   = 20
!  MaxEx- max number of terms in the exact solution
   Integer,Parameter :: MaxEx = 100
!         SPECIFICATIONS FOR LOCAL VARIABLES
   INTEGER i, iDO, j
!
   Real,Parameter :: tMax = 1.0     ! Max Time
!  Alph - the thermal diffusivity
   Real,Parameter :: Alph = 0.1
   Real,Parameter :: Co   = 1.0     ! grad C
   Real,Parameter :: aL   = 4.0     ! wall thickness
   Real,Parameter ::  U   = 0.5     ! velosity
!
   REAL hINIT, t, Tend, tOL, Pi, ajm, aim, dX, x
   Real TSTEP(Nstep), TE(Nx), TD(Nx), XBREAK(Nx), Y(Ldy,Nx), XX(Nx)
   Real*8 sum,avs,rms,dmp
!  If you want to start array visualizer (AV) to put AV_true = .True.
   Logical :: AV_true = .True.
!         Array for Array Viewer
   Real, Allocatable :: UXTEND(:,:)
!dec$attributes array_visualizer :: UXTEND
!
!         SPECIFICATIONS FOR FUNCTIONS
   EXTERNAL FCNBC, FCNUT
!
!   Open output file 'Line_101.rez'
   Open(2,file = 'Line_101.rez')
!
!         Set breakpoints and initial conditions
    Pi = const('pi')
   ajm = Nstep - 1
   aim = Nx - 1
    dX = aL/aim
   Y(1,1:Nx) = 0.0
   Do i = 1,Nx
```

```
                x = FLOAT(i-1)/aim
          XBREAK(i) = x
    If(x .LE. 0.5) Y(1,i) = 1.0
     If(abs(x - 0.5) .LT. 1.0e-4) Y(1,i) = 0.5
   End Do
   Y(1,1) = 1.0 ; Y(1,Nx) = 0.0
!         Set parameters for MOLCH
   tOL = SQRT(AMACH(4))
   hINIT = 0.01*tOL
   t = 0.0
   iDO = 1
!
   allocate(UXTEND(Nstep,Nx))
   UXTEND = 0.0
!
   Do j= 1,Nstep  ! Start Main cycle
!
   Tend = FLOAT(j)/FLOAT(Nstep)
!
!  This puts more output for small t values
!  where action is fastest.
!
   Tend = Tend*Tend
!
!         Solve the problem
   Call MOLCH (iDO, FCNUT, FCNBC, nPDES, t, Tend, Nx, &
     &            XBREAK, tOL, hINIT, Y, Ldy)
!
   If(iDO == 3) Exit
!
   UXTEND(j,:) = Co*Y(1,1:Nx)
      TD(1:Nx) = Co*Y(1,1:Nx)
      TSTEP(j) = t
!         Print results
   If (J .EQ. 1) Then
   Write(2,'(/a)') '#    t          Y '
   Write(2,'(9x,11F9.2)') (XBREAK(i),i=1,Nx,2)
                  End If
    Write(2,'(F9.2,11F9.4)') t,(Y(1,i),i=1,Nx,2)
!
   If(t >= 1.0) Then
!
   Call EXACT
!
   do i = 1,Nx
          dmp = TE(i) - TD(i)
          sum = sum + dmp*dmp
   end do
   Write(2,100) tMax*t,(TD(i),i=1,Nx,2)
100Format(/,' T= ',F7.2,' TD= ',11F7.2)
   Write(2,101) tMax*t,(TE(i),i=1,Nx,2)
101Format(/,' T= ',F7.2,' TE= ',11F7.2,//)
!
!  rms is the RMS error
!
   avs = sum/(1.+ ajm)
   rms = dsqrt(avs)
   Write(2,'(/a,1pD11.4,/)') ' RMS Line = ',rms
!         Final call to release workspace
   iDO = 3
   End If

   End Do
! End Main cycle
```

```
!  To use array visualizer
   If(AV_true) Call v3D(UXTEND,XX,TSTEP,Nstep,Nx)
   deallocate(UXTEND)
!
   Close(2)

   Stop

   Contains

   Subroutine EXACT
!
!  Exact solution for propagating
!  temp-front
!
   Integer i,k,ip,Jm,inc
   Real ai,dxm,dem,dam,sne

   Real,Dimension(Nx) :: X0

       TE = 0.0
    TE(1) = 1.0
   Do i = 1,Nx
            ai = i - 1
     X0(i) = -2.0 + dX*ai
   End Do
            XX = X0
   Do i = 2,Nx-1
       dem = 0.0
      Do k = 1,MaxEx
           ai = 2*k - 1
          dam = ai*Pi/lN
          sne = sin(dam*(X0(i)-U*tMax))
          dxm =- Alph*dam*dam*tMax
          If(dxm .LT. -20.0) exit
          dem = dem + (sne/ai)*exp(dxm)
     End Do
     TE(i) = 0.5 - 2.0*dem/Pi

   End Do
   End Subroutine EXACT
   End Program Line
!
   SUBROUTINE FCNUT (NPDES, X, T, U, UX, UXX, UT)
!
   Implicit none
!
   Real,Parameter :: Alph = 0.1/16.
   Real,Parameter :: c = 0.5/4.
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, U(*), UX(*), UXX(*), UT(*)
!
!         Define the PDE
   UT(1) = - c*UX(1) + Alph*UXX(1)
   RETURN
   END SUBROUTINE FCNUT
!
   SUBROUTINE FCNBC (NPDES, X, T, ALPHA, BETA, GAMP)
!         SPECIFICATIONS FOR ARGUMENTS
   INTEGER NPDES
   REAL X, T, ALPHA(*), BETA(*), GAMP(*)
!
   ALPHA(1) = 1.0
```

```
    BETA(1) = 0.0
    GAMP(1) = 0.0
   RETURN
   END SUBROUTINE FCNBC

   Subroutine v3D(fun,X,T,mValues,nValues)
!         To output array as 3D-figure
   Use avdef
   Use avviewer
   Use dflib
   Integer :: nValues,mValues
   Real    :: fun(mValues,nValues),X(nValues),T(mValues)
   Integer :: hv,status,nError
   Character(1) :: key
   Character(av_max_label_len) :: xLabel = 'X', &
                yLabel = 't / tMax',zLabel = 'T'
!         To take AV name of array
   Call faglStartWatch(fun, status)
!
! Call StartWatch to let the axis-point know we're
!  interested in viewing fun
   Call faglStartWatch(X, status)
   Call faglStartWatch(T, status)
   Write(*,*) " Starting ARRAY Viewer "
!         Start AV by fav-programm
   Call favStartViewer(hv,status)
   If(status /= 0) Then
     Call favGetErrorNo(hv,nError,status)
         If(nError /= 0) Then
   Write(*,*) ' Array Viewer reports error = ',nError
         Stop
         End If
   End If
!         To take AV data
   Call favSetArray(hv,fun,status)
!         To take file name
   Call favSetArrayName(hv,"Line-10.agl",status)
!         To view array as vector graph
   Call favSetGraphType(hv,HeightPlot,status)
!         To set user's procedure named of axis
   Call favSetUseAxisLabel(hv,x_axis,1,status)
   Call favSetUseAxisLabel(hv,y_axis,1,status)
   Call favSetUseAxisLabel(hv,z_axis,1,status)
!         New (instead of "dim1")       name x-axis.
   Call favSetAxisLabel(hv,x_axis,xLabel,status)
!         New (instead of "dim2")       name y-axis.
   Call favSetAxisLabel(hv,y_axis,yLabel,status)
!         New (instead of "z-axis")     name z-axis.
   Call favSetAxisLabel(hv,z_axis,zLabel,status)
!
!         Setup the axis scales.
   Call favSetDimScale(hv, 2, X, status)
    Call favSetDimScale(hv, 1, T, status)
!
!         To display AV at screen
   Call favShowWindow(hv,av_true,status)
   Write(*,*) " Press any key to close down the viewer"
   key = getcharqq()
!         To close AV
   Call favEndViewer(hv,status)
   Call faglEndWatch(fun,status)
   Call faglEndWatch(X, status)
    Call faglEndWatch(T, status)
   End Subroutine v3D
```