Numerical Methods for Ordinary Differential Equations

István Faragó

30.06.2013.

Contents

Introduction, motivation

Applications and modelling and their learning and teaching at schools and universities have become a prominent topic in the last decades in view of the growing worldwide importance of the usage of mathematics in science, technology and everyday life. Given the worldwide impending shortage of youngsters who are interested in mathematics and science it is highly necessary to discuss possibilities to change mathematics education in school and tertiary education towards the inclusion of real world examples and the competencies to use mathematics to solve real world problems.

This book is devoted to the theory and solution of ordinary differential equations. Why is this topic chosen?

In science, engineering, economics, and in most areas where there is a quantitative component, we are greatly interested in describing how systems evolve in time, that is, in describing the dynamics of a system. In the simplest one-dimensional case the state of a system at any time t is denoted by a function, which we generically write as $u = u(t)$. We think of the dependent variable u as the state variable of a system that is varying with time t, which is the independent variable. Thus, knowing u is tantamount to knowing what state the system is in at time t. For example, $u(t)$ could be the population of an animal species in an ecosystem, the concentration of a chemical substance in the blood, the number of infected individuals in a flu epidemic, the current in an electrical circuit, the speed of a spacecraft, the mass of a decaying isotope, or the monthly sales of an advertised item. The knowledge of $u(t)$ for a given system tells us exactly how the state of the system is changing in time. In the literature we always use the variable u for a generic state; but if the state is "population", then we may use p or N; if the state is voltage, we may use V. For mechanical systems we often use $x = x(t)$ for the position. One way to obtain the state $u(t)$ for a given system is to take measurements at different times and fit the data to obtain a nice formula for $u(t)$. Or we might read $u(t)$ off an oscilloscope or some other gauge or monitor. Such curves or formulas may tell us how a system behaves in time, but they do not give us insight into why a system behaves in the way we observe. Therefore we try to for-

mulate explanatory models that underpin the understanding we seek. Often these models are dynamic equations that relate the state $u(t)$ to its rates of change, as expressed by its derivatives $u'(t), u''(t), \ldots$, and so on. Such equations are called differential equations and many laws of nature take the form of such equations. For example, Newtons second law for the motion for a mass acted upon by external forces can be expressed as a differential equation for the unknown position $x = x(t)$ of the mass.

In summary, a differential equation is an equation that describes how a state $u(t)$ changes. A common strategy in science, engineering, economics, etc., is to formulate a basic principle in terms of a differential equation for an unknown state that characterizes a system and then solve the equation to determine the state, thereby determining how the system evolves in time. Since we have no hope of solving the vast majority of differential equations in explicit, analytic form, the design of suitable numerical algorithms for accurately approximating solutions is essential. The ubiquity of differential equations throughout mathematics and its applications has driven the tremendous research effort devoted to numerical solution schemes, some dating back to the beginnings of the calculus. Nowadays, one has the luxury of choosing from a wide range of excellent software packages that provide reliable and accurate results for a broad range of systems, at least for solutions over moderately long time periods. However, all of these packages, and the underlying methods, have their limitations, and it is essential that one be able to recognize when the software is working as advertised, and when it produces spurious results! Here is where the theory, particularly the classification of equilibria and their stability properties, as well as first integrals and Lyapunov functions, can play an essential role. Explicit solutions, when known, can also be used as test cases for tracking the reliability and accuracy of a chosen numerical scheme.

This book is based on the lectures notes given over the past 10+ years, mostly in Applied analysis at Eötvös Loránd University. The course is taken by second- and third-year graduate students from mathematics.

The book is organized into two main parts. Part I deals with initial value problem for first order ordinary differential equations. Part II concerns boundary value problems for second order ordinary differential equations. The emphasis is on building an understanding of the essential ideas that underlie the development, analysis, and practical use of the different methods.

The numerical solution of differential equations is a central activity in science and engineering, and it is absolutely necessary to teach students some aspects of scientific computation as early as possible. I tried to build in flexibility regarding a computer environment. The text allows students to use a

calculator or a computer algebra system to solve some problems numerically and symbolically, and templates of MATLAB programs and commands are given[1].

I feel most fortunate to have had so many people communicate with me regarding their impressions of the topic of this book. I thank the many students and colleagues who have helped me tune these notes over the years. Special thanks go to Ágnes Havasi, Tamás Szabó and Gábor Csörgő for many useful suggestions and careful proofreading of the entire text. As always, any remaining errors are my responsibility.

Budapest, June 2013.

István Faragó

[1]In the textbook we use the name "MATLAB" everywhere. We note that in several places the the notations "Matlab" and "MATLAB® " are also used.

Part I

Numerical methods for initial value problems

Chapter 1

Basics of the theory of initial value problems

Definition 1.0.1. *Let $G \subset \mathbb{R} \times \mathbb{R}^d$ be some given domain (i.e., a connected and open set), $(t_0, \overline{\boldsymbol{u}}_0) \in G$ a given point ($t_0 \in \mathbb{R}$, $\overline{\boldsymbol{u}}_0 \in \mathbb{R}^d$), and $\boldsymbol{f} \colon G \to \mathbb{R}^d$ a given continuous mapping. The problem*

$$\frac{d\boldsymbol{u}(\cdot)}{dt} = \boldsymbol{f}(\cdot, \boldsymbol{u}), \quad \boldsymbol{u}(t_0) = \overline{\boldsymbol{u}}_0 \tag{1.1}$$

is called initial value problem*, or, alternatively,* Cauchy problem.

Let us consider the problem (1.1) coordinate-wise. Denoting by $u_i(\cdot)$ the i-th coordinate-function of the unknown vector-valued function $\mathbf{u}(\cdot)$, by $f_i : G \to \mathbb{R}$ the i-th coordinate-function of the vector-valued function $\mathbf{f}$, and by u_{0i} the i-th coordinate of the vector $\overline{\mathbf{u}}_0$, $(i = 1, 2, \ldots, d)$, we can rewrite the Cauchy problem in the following coordinate-wise form:

$$\begin{aligned} &\frac{du_i(\cdot)}{dt} = f_i(\cdot, u_0, u_1, \ldots, u_d), \\ &u_i(t_0) = u_{0i}, \end{aligned} \tag{1.2}$$

where $i = 1, 2, \ldots, d$.

Solving a Cauchy problem means that we find all functions $\mathbf{u} : \mathbb{R} \to \mathbb{R}^d$ such that in each point of the interval $I \subset \mathbb{R}$ the function can be substituted into problem (1.1), and it also satisfies these equations.

Definition 1.0.2. *A continuously differentiable function $\boldsymbol{u} : I \to \mathbb{R}^d$ (I is an open interval) is called* solution of the Cauchy problem (1.1) *when*

- $\{(t, \boldsymbol{u}(t)) : \quad t \in I\} \subset G$*, and* $t_0 \in I$*;*

- $\dfrac{d\boldsymbol{u}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{u}(t))$ *for all* $t \in I$,
- $\boldsymbol{u}(t_0) = \overline{\boldsymbol{u}}_0$.

When the Cauchy problem (1.1) serves as a mathematical model of a certain real-life (economic, engineering, financial, etc.) phenomenon, then a basic requirement is the existence of a unique solution to the problem. In order to guarantee the existence and uniqueness, we introduce the following notation: $H(t_0, \overline{\mathbf{u}}_0) = \{(t, \mathbf{u}) : |t - t_0| \leq \alpha, \|\mathbf{u} - \overline{\mathbf{u}}_0\| \leq \beta\} \subset G$. (This means that $H(t_0, \overline{\mathbf{u}}_0)$ is a closed rectangle of dimension $d+1$ with center at $(t_0, \overline{\mathbf{u}}_0)$.) Since $\mathbf{f}$ is continuous on the closed set $H(t_0, \overline{\mathbf{u}}_0)$, therefore we can introduce the notation for the real number M defined as $M = \max_{H(t_0, \overline{\mathbf{u}}_0)} \|\mathbf{f}(t, \mathbf{u})\|$. Then for any t such that $|t - t_0| \leq \min\{\alpha, \beta/M\}$, there exists a unique solution $\mathbf{u}(t)$ of the Cauchy problem (1.1). Moreover, when the function $\mathbf{f}$ satisfies the *Lipschitz condition in its second variable* on the set $H(t_0, \overline{\mathbf{u}}_0)$, i.e., there exists some constant $L > 0$ such that for any $(t, \mathbf{u}_1), (t, \mathbf{u}_2) \in H(t_0, \overline{\mathbf{u}}_0)$ the relation

$$\|\mathbf{f}(t, \overline{\mathbf{u}}_1) - \mathbf{f}(t, \overline{\mathbf{u}}_2)\| \leq L\|\overline{\mathbf{u}}_1 - \overline{\mathbf{u}}_2\| \tag{1.3}$$

(the so-called *Lipschitz condition*) holds, then the solution is unique.

In the sequel, in the Cauchy problem 1.1 we assume that there exists a subset $H(t_0, \overline{\mathbf{u}}_0) \subset G$ on which $\mathbf{f}$ is continuous and satisfies (in its second variable) the Lipschitz condition. This means that there exists a unique solution on the interval $I_0 := \{t \in I : |t - t_0| \leq T\}$, where $T = \min\{\alpha, \beta/M\}$.

Since t denotes the time-variable, therefore the solution of a Cauchy problem describes the change in time of the considered system. In the analysis of real-life problems we aim to know the development in time of the system, i.e., having information on the position (state) of the system at some initial time-point, how is the system developing? Mathematically this yields that knowing the value of the function $\mathbf{u}(t)$ at $t = t_0$, we want to know its values for $t > t_0$, too. The point $t = t_0$ is called *starting point*, and the given value of the solution at this point is called *initial value*. In general, without loss of generality we may assume that $t_0 = 0$. Therefore, the domain of definition of the solution of problem (1.1) is the interval $[0, T] \subset I$, and hence our Cauchy problem has the following form:

$$\frac{d\mathbf{u}(t)}{dt} = \mathbf{f}(t, \mathbf{u}(t)), \quad t \in [0, T], \tag{1.4}$$

$$\mathbf{u}(0) = \overline{\mathbf{u}}_0. \tag{1.5}$$

Our aim is to define the unknown function $\mathbf{u}(t)$ in this problem.

Remark 1.0.1. Under the assumption of continuity of $\mathbf{f}$,(i.e., $\mathbf{f} \in C(H)$), the solution of the Cauchy problem (1.4)-(1.5) $\mathbf{u}(t)$ is also continuously differentiable, i.e., $\mathbf{u} \in C^1[0,T]$. At the same time, if $\mathbf{f}$ is smoother, then the solution is also smoother: when $\mathbf{f} \in C^p(H)$, then $\mathbf{u} \in C^{p+1}[0,T]$, where $p \in \mathbb{N}$. Hence, by suitably choosing the smoothness of $\mathbf{f}$ we can always guarantee some prescribed smoothness of the solution. Therefore, it means no constraints further on if we assume that the solution is *sufficiently smooth*.

For simplicity, in general we will investigate the numerical methods for the scalar case, where $d = 1$. Then the formulation of the problem is as follows.

Let $Q_T := [0,T] \times \mathbb{R} \subset \mathbb{R}^2$, $f : Q_T \to \mathbb{R}$. The problem of the form

$$\frac{du}{dt} = f(t,u), \quad u(0) = u_0 \tag{1.6}$$

will be called Cauchy problem. (We do not emphasize the scalar property.) We always assume that for the given function $f \in C(Q_T)$ the Lipschitz condition

$$|f(t,u_1) - f(t,u_2)| \leq L\,|u_1 - u_2|\,, \quad \forall (t,u_1),(t,u_2) \in Q_T, \tag{1.7}$$

is satisfied. Moreover, $u_0 \in \mathbb{R}$ is a given number. Hence, our task is to find a sufficiently smooth function $u : [0,T] \to \mathbb{R}$ such that the relations

$$\frac{du(t)}{dt} = f(t,u(t)), \quad \forall t \in [0,T], \quad u(0) = u_0 \tag{1.8}$$

hold.

Remark 1.0.2. Let $g : \mathbb{R}^2 \to \mathbb{R}$ some given function. Is there any connection between its above mentioned two properties, namely, between the continuity and the Lipschitz property w.r.t. the second variable? The answer to this question is negative, as it is demonstrated on the following examples.

- Let $g(x,y) = y^2$. This function is obviously continuous on $G = \mathbb{R}^2$, but the Lipschitz condition (1.7) on G cannot be satisfied for this function. (Otherwise, due to the relation
$$|g(x,y_1) - g(x,y_2)| = \left|y_1^2 - y_2^2\right| = |y_1 + y_2|\,|y_1 - y_2|\,,$$
the expression $|y_1 + y_2|$ would be bounded for any $y_1, y_2 \in \mathbb{R}$, which is a contradiction.)

- Let $g(x,y) = D(x)y$, where $D(x)$ denotes the well-known Dirichlet function. [1] Then g is nowhere continuous, however, due to the relation
$$|g(x,y_1) - g(x,y_2)| = |D(x)|\,|y_1 - y_2| \leq |y_1 - y_2|\,,$$
it satisfies the Lipschitz condition (1.7) with $L = 1$ on $G = \mathbb{R}^2$.

[1] The Dirichlet function is defined as follows: $D(x) = 1$ if x is rational, and $D(x) = 0$ if x is irrational. This function is discontinuous at each point.

Remark 1.0.3. How can the Lipschitz property be guaranteed? Assume that the function $g : \mathbb{R}^2 \to \mathbb{R}$ has uniformly bounded partial derivatives on some subset H_g. Then, using Lagrange's theorem, there exists some $\tilde{y} \in (y_1, y_2)$ such that the relation $g(x, y_1) - g(x, y_2) = \partial_2 g(x, \tilde{y})(y_1 - y_2)$ holds. This implies the validity of the relation (1.7) with Lipschitz constant $L = sup_{H_g} (|\partial_2 g(x, y)|) < \infty$.

Corollary 1.0.3. As a consequence of Remark 1.0.3, we have the following. When the function f in the Cauchy problem (1.8) is continuous on Q_T and it has bounded partial derivatives w.r.t. its second variable, then the Cauchy problem has a unique solution on the interval $[0, T]$.

Chapter 2

An introduction to one-step numerical methods

The theorems considered in Chapter 1 inform us about the existence and uniqueness of the solution of the initial value problem, but there is no answer to the question of how to find its solution. In fact, the solution of such problems in analytical form can be given in very limited cases, only for some rather special right-hand side functions f. Therefore, we define the solution only approximately, which means that — using some suitably chosen *numerical method*, (which consists of a finite number of steps) — we approximate the unknown solution at certain points of the time interval $[0, T]$, where the solution exists. Our aim is to define these numerical methods, i.e., to give the description of those methods (algorithms) which allow us to compute the approximation in the above mentioned points.
In the following our aim is the numerical solution of the problem

$$\frac{du}{dt} = f(t, u), \quad t \in [0, T], \tag{2.1}$$

$$u(0) = u_0, \tag{2.2}$$

where $T > 0$ is such that the initial value problem (2.1)–(2.2) has a unique solution on the interval $[0, T]$. This means that we want to approximate the solution of this problem at a finite number of points of the interval $[0, T]$, denoted by $\{t_0 < t_1 < \cdots < t_N\}$. [1] In the sequel we consider those methods where the value of the approximation at a given time-point t_n is defined only by the approximation at the time-point t_{n-1}. Such methods are called *one-step methods.*

[1] We mention that, based on these approximate values, using some interpolation method we can define some approximation at any point of the interval $[0, T]$.

2.1 The Taylor method

This is one of the oldest methods. By definition, the solution $u(t)$ of the Cauchy problem satisfies the equation (2.1), which results in the equality

$$u'(t) = f(t, u(t)), \quad t \in [0, T]. \tag{2.3}$$

We assume that f is an analytical function, therefore it has partial derivatives of any order on the set Q_T.[5, 11]. Hence, by using the chain rule, by differentiation of the identity (2.3), at some point $t^\star \in [0, T]$ we get the relation

$$\begin{aligned} u'(t^\star) &= f(t^\star, u(t^\star)), \\ u''(t^\star) &= \partial_1 f(t^\star, u(t^\star)) + \partial_2 f(t^\star, u(t^\star))\ u'(t^\star), \\ u'''(t^\star) &= \partial_{11} f(t^\star, u(t^\star)) + 2\partial_{12} f(t^\star, u(t^\star))\ u'(t^\star) + \partial_{22} f(t^\star, u(t^\star))\ (u'(t^\star))^2 + \\ &\quad + \partial_2 f(t^\star, u(t^\star))\ u''(t^\star). \end{aligned} \tag{2.4}$$

Let us notice that knowing the value $u(t^\star)$ all derivatives can be computed exactly.

We remark that theoretically any higher order derivative can be computed in the same way, however, the corresponding formulas become increasingly complicated.

Let $t > t^\star$ such that $[t^\star, t] \subset [0, T]$. Since the solution $u(t)$ is analytical, therefore its Taylor series is reproducing locally this function in some neighbourhood of the point $t^\star$. Hence the Taylor polynomial

$$\sum_{k=0}^{n} \frac{u^{(k)}(t^\star)}{k!}(t - t^\star)^k \tag{2.5}$$

tends to $u(t)$, when t is approaching $t^\star$. Therefore, inside the domain of convergence, the relation

$$u(t) = \sum_{k=0}^{\infty} \frac{u^{(k)}(t^\star)}{k!}(t - t^\star)^k \tag{2.6}$$

holds.

However, we emphasize that the representation of the solution in the form (2.6) is practically not realistic: it assumes the knowledge of partial derivatives of any order of the function f, moreover, to compute the exact value of the solution at some fixed point, this formula requires the summation of an *infinite series*, which is typically not possible.

Hence, the computation of the exact value $u(t)$ by the formula (2.6) is not possible. Therefore we aim to define only its *approximation*. The most natural

idea is to replace the infinite series with the *truncated finite sum*, i.e., the approximation is the p-th order Taylor polynomial of the form

$$u(t) \simeq \sum_{k=0}^{p} \frac{u^{(k)}(t^\star)}{k!}(t-t^\star)^k =: T_{p,u}(t), \tag{2.7}$$

and then the neglected part (the error) is of the order $\mathcal{O}((t-t^\star)^{p+1})$. By the definition, in this approach $T_{p,u}(t)$ yields the Taylor polynomial of the function $u(t)$ at the point $t^\star$.

Based on the formulas (2.7) and (2.4), the following numerical methods can be defined.

a) Taylor method

Let us select $t^\star = 0$, where the initial condition is given.[2]

Then the value $u(t^\star) = u(0)$ is known from the initial condition, and, based on the formula (2.4), the derivatives can be computed *exactly* at this point. Hence, using the approximation (2.7), we have

$$u(t) \simeq \sum_{k=0}^{p} \frac{u^{(k)}(0)}{k!}t^k, \tag{2.8}$$

where, based on (2.4), the values $u^{(k)}(0)$ can be computed.

b) Local Taylor method

We consider the following algorithm.

1. On the interval $[0,T]$ we define the points $t_0, t_1, \dots t_N$, which define the *mesh* $\omega_h := \{0 = t_0 < t_1 < \dots < t_{N-1} < t_N = T\}$. The distances between two neighbouring mesh-points, i.e., the values $h_i = t_{i+1} - t_i$, (where $i = 0, 1, \dots N-1$,) are called *step-size*, while $h = \max_i h_i$ denotes the measure of the mesh. (In the sequel, we define the approximation at the mesh-points, and the approximations to the exact values $u(t_i)$ will be denoted by y_i, while the approximations to the k-th derivatives $u^{(k)}(t_i)$ will be denoted by $y_i^{(k)}$, where $k = 0, 1, \dots, p$.[3]
2. The values $y_0^{(k)}$ for $k = 0, 1, \dots, p$ can be defined *exactly* from the formula (2.4), by substituting $t^\star = 0$.

[2] According to Section 1, the derivatives do exist at the point $t = 0$.

[3] As usual, the zero-th derivative ($k = 0$) denotes the function.

3. Then, according to the formula

$$y_1 = \sum_{k=0}^{p} \frac{y_0^{(k)}}{k!} h_0^k, \tag{2.9}$$

we define the approximation to $u(t_1)$.

4. For $i = 1, 2, \ldots, N-1$, using the values y_i, by (2.4) we define *approximately* $y_i^{(k)}$ (for $k = 0, 1, \ldots, p$), by the substitution $t^\star = t_i$ and $u(t^\star) = u(t_i) \approx y_i$.
5. Using the formula

$$y_{i+1} = \sum_{k=0}^{p} \frac{y_i^{(k)}}{k!} h_i^k, \tag{2.10}$$

we define the approximation to $u(t_{i+1})$.

Using (2.10), let us define the algorithm of the local Taylor method for the special cases $p = 0, 1, 2$!

- For $p = 0$, $y_i = y_0$ for each value of i. Therefore this case is not interesting, and we will not investigate it.
- For $p = 1$ we have

$$y_{i+1} = y_i + y_i' h_i = y_i + h_i f(t_i, y_i), \quad i = 0, 1, \ldots N-1, \tag{2.11}$$

where $y_0 = u_0$ is given.

- For $p = 2$ we have

$$y_{i+1} = y_i + h_i y_i' + \frac{h_i^2}{2} y_i'' = y_i + h_i f(t_i, y_i) + \frac{h_i^2}{2} \left(\partial_1 f(t_i, y_i) + \partial_2 f(t_i, y_i) f(t_i, y_i) \right), \tag{2.12}$$

where $i = 0, 1, \ldots N-1$, and y_0 is given.

Let us compare the above methods.

1. In both cases we use the Taylor polynomial of order p, therefore both methods require the knowledge of all partial derivatives of f, up to order $p-1$. This means the computation of $p(p-1)/2$ partial derivatives, and for each it is necessary to evaluate the functions, too. This results in huge computational costs, even for moderate values of p. Therefore, in practice the value p is chosen to be small.[4] This results in the fact that the accuracy of the Taylor method is significantly limited in the applications.

[4] In the last period the spacial program tools called *symbolic computations* give possibility for the computations of the derivatives automatically, however, the above problem still exists.

2. The convergence of the Taylor method by increase of p is theoretically shown only for those values which are in the convergence interval of the Taylor series. This is one of the most serious disadvantage of the method: the convergence radius of the solution is usually unknown.

3. When we want to get the approximation only at one point $t = \hat{t}$, and this point is inside the convergence domain, then the Taylor method is beneficial, because the approximation can be obtained in one step. The local Taylor method avoids the above shortcoming: by choosing the step-size h to be sufficiently small we remain inside the convergence domain. However, in this case we should solve n problems, where $h_0 + h_1 + \ldots + h_{n-1} = \hat{t}$, since we can give the solution only on the complete time interval $[0, \hat{t}]$.

4. For the Taylor method the error (difference between the exact and the numerical solution) can be defined by the Lagrange error formula for the Taylor polynomial. However, this is not possible for the local Taylor method, because the error consist of two parts:

   a) at each step there is the same error as for the Taylor method, which arises from the replacement of the function with its Taylor polynomial of order n;

   b) the coefficients of the Taylor polynomial, i.e., the derivatives of the solution are computed only approximately, with some error. (During the computation these errors can grow up.)

5. We note that for the construction of the numerical method it is not necessary to require that the solution is analytical: it is enough to assume that the solution is $p + 1$ times continuously differentiable, i.e., $f \in C^p(Q_T)$.

Example 2.1.1. *We consider the Cauchy problem*

$$\begin{aligned} u' &= -u + t + 1, \quad t \in [0, 1], \\ u(0) &= 1. \end{aligned} \tag{2.13}$$

The exact solution is $u(t) = \exp(-t) + t$.

In this problem $f(t, u) = -u + t + 1$, therefore

$$\begin{aligned} u'(t) &= -u(t) + t + 1, \\ u''(t) &= -u'(t) + 1 = u(t) - t, \\ un'''(t) &= -u(t) + t, \end{aligned} \tag{2.14}$$

i.e., $u(0) = 1, \;\; u'(0) = 0, \;\; u''(0) = 1, \;\; u'''(0) = -1$. The global Taylor method results in the following approximation polynomials:

$$\begin{aligned} T_{1,u}(t) &= 1, \\ T_{2,u}(t) &= 1 + t^2/2, \\ T_{3,u}(t) &= 1 + t^2/2 - t^3/6. \end{aligned} \tag{2.15}$$

Hence, at the point $t = 1$ we have $T_{1,u}(1) = 1$, $T_{2,u}(1) = 1.5$, $T_{3,u}(1) = 1.333$). (We can also easily define the values $T_{4,u}(1) = 1.375$ and $T_{5,u}(1) = 1.3666$.) As we can see, these values approximate the value of the exact solution $u(1) = 1.367879$ only for larger values of n.

Let us apply now the local Taylor method taking into account the derivatives under (2.14). The algorithm of the first order method is

$$y_{i+1} = y_i + h_i(-y_i + t_i + 1), \quad i = 0, 1, \ldots, N-1, \tag{2.16}$$

while the algorithm of the second order method is

$$y_{i+1} = y_i + h_i(-y_i + t_i + 1) + \frac{h_i^2}{2}(y_i - t_i), \quad i = 0, 1, \ldots, N-1,$$

where $h_1 + h_2 + \ldots + h_N = T$. In our computations we have used the step-size $h_i = h = 0.1$. In Table 2.1 we compared the results of the global and local Taylor methods at the mesh-point of the interval $[0, 1]$. (LT1 and LT2 mean the first and second order local Taylor method, while T1, T2 and T3 are the first, second and third order Taylor methods, respectively.)

Using some numerical method, we can define a numerical solution at the mesh-points of the grid. Comparing the numerical solution with the exact solution, we define the error function, which is a grid function on the mesh on which the numerical method is applied. This error function (which is a vector) can be characterized by the maximum norm. In Table 2.2 we give the magnitude of the maximum norm of the error function on the meshes for decreasing step-sizes. We can observe that by decreasing h the maximum norm is strictly decreasing for the local Taylor method, while for the global Taylor method the norm does not change. (This is a direct consequence of the fact that the global Taylor method is independent of the mesh-size.)

The local Taylor method is a so-called *one-step method* (or, alternatively, a *two-level method*). This means that the approximation at the time level $t = t_{i+1}$ is defined with the approximation obtained at the time level $t = t_i$ only. The error analysis is rather complicated. As the above example shows, the difference between the exact solution $u(t_{i+1})$ and the numerical solution y_{i+1} is caused by several reasons.

t_i	the exact solution	LT1	LT2	T1	T2	T3
0.1	1.0048	1.0000	1.0050	1.0000	1.0050	1.0048
0.2	1.0187	1.0100	1.0190	1.0000	1.0200	1.0187
0.3	1.0408	1.0290	1.0412	1.0000	1.0450	1.0405
0.4	1.0703	1.0561	1.0708	1.0000	1.0800	1.0693
0.5	1.1065	1.0905	1.1071	1.0000	1.1250	1.1042
0.6	1.1488	1.1314	1.1494	1.0000	1.1800	1.1440
0.7	1.1966	1.1783	1.1972	1.0000	1.2450	1.1878
0.8	1.2493	1.2305	1.2500	1.0000	1.3200	1.2347
0.9	1.3066	1.2874	1.3072	1.0000	1.4050	1.2835
1.0	1.3679	1.3487	1.3685	1.0000	1.5000	1.3333

Table 2.1: Comparison of the local and global Taylor methods on the mesh with mesh-size $h = 0.1$.

mesh-size	LT1	LT2	T1	T2	T3
0.1	$1.92e-02$	$6.62e-04$	0.3679	0.1321	0.0345
0.01	$1.80e-03$	$6.12e-06$	0.3679	0.1321	0.0345
0.001	$1.85e-04$	$6.14e-08$	0.3679	0.1321	0.0345
0.0001	$1.84e-05$	$6.13e-10$	0.3679	0.1321	0.0345

Table 2.2: Maximum norm errors for the local and global Taylor methods for decreasing mesh-size h.

- The first reason is the *local truncation error*, which is due to the replacement of the Taylor series by the Taylor polynomial, assuming that we know the exact value at the point $t = t_i$. The order of the difference on the interval $[t_i, t_i + h_i]$, i.e., the order of magnitude of the expression $u(t) - T_{n,u}(t)$ defines the *order of the local error*. When this expression has the order $\mathcal{O}(h_i^{p+1})$, then the method is called p-th order.

- In each step (except for the first step) of the construction, instead of the exact values their approximations are included. The effect of this inaccuracy may be very significant and they can extremely accumulate during the computation (this is the so-called instability).

- In the computational process we have also *round-off error*, also called *rounding error*, which is the difference between the calculated approximation of a number and its exact mathematical value. Numerical analysis specifically tries to estimate this error when using approximation equations and/or algorithms, especially when using finitely many digits to represent real numbers (which in theory have infinitely many digits). In our work we did not consider the round-off error, which is always present in computer calculations.[5]

- When we solve our problem on the interval $[0, t^\star]$, then we consider the difference between the exact solution and the numerical solution at the point $t^\star$. We analyze the error which arises due to the first two sources, and it is called *global error*. Intuitively, we say that some method is convergent at some fixed point $t = t^\star$ when by approaching zero with the maximum step-size of the mesh the global error at this point tends to zero. The order of the convergence of this limit to zero is called *order of convergence* of the method. This order is independent of the round-off error. In the numerical computations, to define the approximation at the point $t = t^\star$, we have to execute approximately n steps, where $nh = t^\star$. Therefore, in case of local truncation error of the order $\mathcal{O}(h^{p+1})$, the expected magnitude of the global error is $\mathcal{O}(h^p)$. In Table 2.2 the results for the methods LT1 and LT2 confirm this conjecture: method LT1 is convergent in the first order, while method LT2 in the second order at the point $t^\star = 1$.

The nature of the Taylor method for the differential equation $u' = 1 - t\sqrt[3]{u}$ can be observed on the link

[5] At the present time there is no universally accepted method to analyze the round-off error after a large number of time steps. The three main methods for analyzing round-off accumulation are the analytical method, the probabilistic method and the interval arithmetic method, each of which has both advantages and disadvantages.

http://math.fullerton.edu/mathews/a2001/Animations/OrdinaryDE/Taylor/Taylor.html

2.2 Some simple one-step methods

In the previous section we saw that the local Taylor method, especially for $p = 1$ is beneficial: for the computation by the formula (2.11) the knowledge of the partial derivatives of the function f is not necessary, and by decreasing the step-size of the mesh the unknown exact solution is well approximated at the mesh-points. Our aim is to define further one-step methods having similar good properties.

The LT1 method was obtained by the approximation of the solution on the subinterval $[t_i, t_{i+1}]$ by its first order Taylor polynomial.[6] Then the error (the local truncation error) is

$$|u(t_{i+1}) - T_{1,u}(t_{i+1})| = \mathcal{O}(h_i^2), \quad i = 0, 1, \ldots, N-1, \tag{2.17}$$

which means that the approximation is exact in the second order. Let us define instead of $T_{1,u}(t)$ some other, first order polynomial $P_1(t)$, for which the estimate (2.17) remains true, i.e., the estimation

$$|u(t_{i+1}) - P_1(t_{i+1})| = \mathcal{O}(h_i^2) \tag{2.18}$$

holds.

The polynomial $T_{1,u}(t)$ is the tangent line at the point $(t_i, u(t_i))$ to the exact solution. Therefore, we seek such a first order polynomial $P_1(t)$, which passes through this point, but whose direction is defined by the tangent lines to the solution $u(t)$ at the points t_i and t_{i+1}. To this aim, let $P_1(t)$ have the form $P_1(t) := u(t_i) + \alpha(t - t_i)$ $(t \in [t_i, t_{i+1}])$, where $\alpha = \alpha(u'(t_i), u'(t_{i+1}))$. (E.g., by the choice $\alpha = u'(t_i)$ we get $P_1(t) = T_{1,u}(t)$, and then the estimation (2.18) holds.)

Is any other suitable choice of α possible? Since

$$u(t_{i+1}) = u(t_i) + u'(t_i)h_i + \mathcal{O}(h_i^2), \tag{2.19}$$

therefore

$$u(t_{i+1}) - P_1(t_{i+1}) = h_i(u'(t_i) - \alpha) + \mathcal{O}(h_i^2),$$

i.e., the relation (2.18) is satisfied if and only if the estimation

$$\alpha - u'(t_i) = \mathcal{O}(h_i) \tag{2.20}$$

holds.

[6] In each subinterval $[t_i, t_{i+1}]$ we define a different Taylor polynomial of the first order, but the dependence of the polynomial on the index i will not be denoted.

Theorem 2.2.1. *For any $\theta \in \mathbb{R}$, under the choice of α by*

$$\alpha = (1-\theta)u'(t_i) + \theta u'(t_{i+1}) \tag{2.21}$$

the estimation (2.20) *is true.*

Proof. Let us apply (2.19) to the function $u'(t)$. Then we have

$$u'(t_{i+1}) = u'(t_i) + u''(t_i)h_i + \mathcal{O}(h_i^2). \tag{2.22}$$

Substituting the relation (2.22) into the formula (2.21), we get

$$\alpha - u'(t_i) = \theta u''(t_i)h_i + \mathcal{O}(h_i^2), \tag{2.23}$$

which proves the statement. □

Corollary 2.2.2. The above polynomial $P_1(t)$ defines the one-step numerical method of the form

$$y_{i+1} = y_i + \alpha h_i, \tag{2.24}$$

where, based on the relations (2.21) and (2.1), we have

$$\alpha = (1-\theta)f(t_i, y_i) + \theta f(t_{i+1}, y_{i+1}). \tag{2.25}$$

Definition 2.2.3. *The numerical method defined by* (2.24)-(2.25) *is called θ-method.*

Remark 2.2.1. As for any numerical method, for the θ-method we also assume that y_i is an approximation to the exact value $u(t_i)$, and the difference between the approximation and the exact value originates – as for the Taylor method – from two sources

a) at each step we replace the exact solution function $u(t)$ by the first order polynomial $P_1(t)$,

b) in the polynomial $P_1(t)$ the coefficient α (i.e., the derivatives of the solution function) can be defined only approximately.

Since the direction α is defined by the derivatives of the solution at the points t_i and t_{i+1}, therefore we select its value to be between these two values. This requirement implies that the parameter θ is chosen from the interval $[0, 1]$. In the sequel, we consider three special values of $\theta \in [0, 1]$ in more detail.

2.2.1 Explicit Euler method

Let us consider the θ-method with the choice $\theta = 0$. Then the formulas (2.24) and (2.25) result in the following method:

$$y_{i+1} = y_i + h_i f(t_i, y_i), \quad i = 0, 1, \ldots, N-1. \tag{2.26}$$

Since y_i is the approximation of the unknown solution $u(t)$ at the point $t = t_i$, therefore

$$y_0 = u(0) = u_0, \tag{2.27}$$

i.e., in the iteration (2.26) the starting value y_0, corresponding to $i = 0$, is given.

Definition 2.2.4. *The one-step method* (2.26)–(2.27) *is called explicit Euler method.*[7] *(Alternatively, it is also called forward Euler method.)*

In case $\theta = 0$ we have $\alpha = u'(t_i)$, therefore the polynomial P_1 (which defines the method) coincides with the first order Taylor polynomial. Therefore the explicit Euler method is the same as the local Taylor method of the first order, defined in (2.11).

Remark 2.2.2. The method (2.26)–(2.27) is called explicit, because the approximation at the point t_{i+1} is defined directly from the approximation, given at the point t_i.

We can characterize the explicit Euler method (2.26)–(2.27) on the following example, which gives a good insight of the method.

Example 2.2.5. *The simplest initial value problem is*

$$u' = u, \quad u(0) = 1, \tag{2.28}$$

whose solution is, of course, the exponential function $u(t) = e^t$.

[7] Leonhard Euler (1707–1783) was a pioneering Swiss mathematician and physicist. He made important discoveries in fields as diverse as infinitesimal calculus and graph theory. He introduced much of the modern mathematical terminology and notation, particularly for mathematical analysis, such as the notion of a mathematical function. He is also renowned for his work in mechanics, fluid dynamics, optics, and astronomy. Euler spent most of his adult life in St. Petersburg, Russia, and in Berlin, Prussia. He is considered to be the preeminent mathematician of the 18th century, and one of the greatest of all time. He is also one of the most prolific mathematicians ever; his collected works fill between 60 and 80 quarto volumes. A statement attributed to Pierre-Simon Laplace expresses Euler's influence on mathematics: "Read Euler, read Euler, he is the master of us all."

Since for this problem $f(t,u) = u$, the explicit Euler method with a fixed step size $h > 0$ takes the form

$$y_{i+1} = y_i + hy_i = (1+h)y_i.$$

This is a linear iterative equation, and hence it is easy to get

$$y_i = (1+h)^i u_0 = (1+h)^i.$$

Then this is the proposed approximation to the solution $u(t_i) = e^{t_i}$ at the mesh point $t_i = ih$. Therefore, when using the Euler scheme to solve the differential equation, we are effectively approximating the exponential by a power function

$$e^{t_i} = e^{ih} \approx (1+h)^i.$$

When we use simply $t^\star$ to indicate the fixed mesh-point $t_i = ih$, we recover, in the limit, a well-known calculus formula:

$$e^{t^\star} = \lim_{h\to 0}(1+h)^{t^\star/h} = \lim_{i\to\infty}(1+t^\star/h)^i.$$

A reader familiar with the computation of compound interest will recognize this particular approximation. As the time interval of compounding, h, gets smaller and smaller, the amount in the savings account approaches an exponential.

In Remark 2.2.1 we listed the sources of the error of a numerical method. A basic question is the following: by refining the mesh what is the behavior of the numerical solution at some fixed point $t^\star \in [0,T]$? More precisely, we wonder whether by increasing the step-size of the mesh to zero the difference of the numerical solution and the exact solution tends to zero. In the sequel, we consider this question for the explicit Euler method. (As before, we assume that the function f satisfies a Lipschitz condition in its second variable, and the solution is sufficiently smooth.)

First we analyze the question on a sequence of refined uniform meshes. Let

$$\omega_h := \{t_i = ih;\ i = 0,1,\ldots,N;\ h = T/N\}$$

($h \to 0$) be given meshes and assume that $t^\star \in [0,T]$ is such a fixed point which belongs to each mesh. Let n denote on a fixed mesh ω_h the index for which $nh = t^\star$. (Clearly, n depends on h, and in case $h \to 0$ the value of n tends to infinity.) We introduce the notation

$$e_i = y_i - u(t_i),\ i = 0,1,\ldots,N \tag{2.29}$$

for the global error at some mesh-point t_i. In the sequel we analyze e_n by decreasing h, i.e., we analyze the difference between the exact and the numerical solution at the fixed point $t^\star$ for $h \to 0$.[8] From the definition of the global error (2.29) obviously we have $y_i = e_i + u(t_i)$. Substituting this expression into the formula of the explicit Euler method of the form (2.26), we get the relation

$$\begin{aligned} e_{i+1} - e_i &= -(u(t_{i+1}) - u(t_i)) + hf(t_i, e_i + u(t_i)) \\ &= [hf(t_i, u(t_i)) - (u(t_{i+1}) - u(t_i))] \\ &+ h\left[f(t_i, e_i + u(t_i)) - f(t_i, u(t_i))\right]. \end{aligned} \tag{2.30}$$

Let us introduce the notations

$$\begin{aligned} g_i &= hf(t_i, u(t_i)) - (u(t_{i+1}) - u(t_i)), \\ \psi_i &= f(t_i, e_i + u(t_i)) - f(t_i, u(t_i)). \end{aligned} \tag{2.31}$$

Hence we get the relation

$$e_{i+1} - e_i = g_i + h\psi_i, \tag{2.32}$$

which is called *error equation of the explicit Euler method*.

Remark 2.2.3. Let us briefly analyze the two expressions in the notations (2.31). The expression g_i shows how exactly the solution of the differential equation satisfies the formula of the explicit Euler method (2.26), written in the form $hf(t_i, y_i) - (y_{i+1} - y_i) = 0$. This term is present due to the replacement of the solution function $u(t)$ on the interval $[t_i, t_{i+1}]$ by the first order Taylor polynomial. The second expression ψ_i characterizes the magnitude of the error, arising in the formula of the method for the computation y_{i+1}, when we replace the exact (and unknown) value $u(t_i)$ by its approximation y_i.

Due to the Lipschitz property, we have

$$|\psi_i| = |f(t_i, e_i + u(t_i)) - f(t_i, u(t_i))| \le L|(e_i + u(t_i)) - u(t_i)| = L|e_i|. \tag{2.33}$$

Hence, based on (2.32) and (2.33), we get

$$|e_{i+1}| \le |e_i| + |g_i| + h|\psi_i| \le (1 + hL)|e_i| + |g_i| \tag{2.34}$$

[8] Intuitively it is clear that the condition $t^\star \in \omega_h$ for any $h > 0$ can be relaxed: it is enough to assume that the sequence of mesh-points (t_n) is convergent to the fixed point $t^\star$, i.e., the condition $\lim_{h\to 0}(t^\star - t_n) = 0$ holds.

for any $i = 0, 1, \ldots, n-1$. Using this relation, we can write the following estimation for the global error e_n:

$$\begin{aligned}
|e_n| &\leq (1+hL)|e_{n-1}| + |g_{n-1}| \leq (1+hL)\left[(1+hL)|e_{n-2}| + |g_{n-2}|\right] + |g_{n-1}| \\
&= (1+hL)^2|e_{n-2}| + \left[(1+hL)|g_{n-2}| + |g_{n-1}|\right] \leq \ldots \\
&\leq (1+hL)^n|e_0| + \sum_{i=0}^{n-1}(1+hL)^i|g_{n-1-i}| \\
&< (1+hL)^n\left[|e_0| + \sum_{i=0}^{n-1}|g_{n-1-i}|\right].
\end{aligned} \tag{2.35}$$

(In the last step we used the obvious estimation $(1+hL)^i < (1+hL)^n$, $i = 0, 1, \ldots, n-1$.) Since for any $x > 0$ the inequality $1 + x < \exp(x)$ holds, therefore, by use of the equality $nh = t^\star$, we have$(1+hL)^n < \exp(nhL) = \exp(Lt^\star)$. Hence, based on (2.35), we get

$$|e_n| \leq \exp(Lt^\star)\left[|e_0| + \sum_{i=0}^{n-1}|g_{n-1-i}|\right]. \tag{2.36}$$

Let us give an estimation for $|g_i|$. One can easily see that the equality

$$u(t_{i+1}) - u(t_i) = u(t_i + h) - u(t_i) = hu'(t_i) + \frac{1}{2}u''(\xi_i)h^2 \tag{2.37}$$

is true, where $\xi_i \in (t_i, t_{i+1})$ is some fixed point. Since $f(t_i, u(t_i)) = u'(t_i)$, therefore, according to the definition of g_i in (2.31), the inequality

$$|g_i| \leq \frac{M_2}{2}h^2, \quad M_2 = \max_{[0,t^\star]}|u''(t)| \tag{2.38}$$

holds. Using the estimations (2.36) and (2.38), we get

$$|e_n| \leq \exp(Lt^\star)\left[|e_0| + hn\frac{M_2}{2}h\right] = \exp(Lt^\star)\left[|e_0| + \frac{t^\star M_2}{2}h\right]. \tag{2.39}$$

Since $e_0 = 0$, therefore,

$$|e_n| \leq \exp(Lt^\star)\frac{t^\star M_2}{2}h. \tag{2.40}$$

Hence, in case $h \to 0$ we have $e_n \to 0$, moreover, $e_n = \mathcal{O}(h)$.

The convergence of the explicit Euler method on some suitably chosen, non-uniform mesh can be shown, too. Further we will show it. Let us consider the sequence of refined meshes

$$\omega_{h_v} := \{0 = t_0 < t_1 < \ldots < t_{N-1} < t_N = T\}.$$

We use the notations $h_i = t_{i+1} - t_i$, $i = 0, 1, \ldots, N-1$ and $h = T/N$. In the sequel we assume that with increasing the number of mesh-points the grid becomes finer everywhere, i.e., there exists a constant $0 < c < \infty$ such that

$$h_i \leq ch, \quad i = 1, 2, \ldots, N \tag{2.41}$$

for any N. We assume again that the fixed point $t^\star \in [0, T]$ is an element of each mesh. As before, on some fixed mesh n denotes the index for which $h_0 + h_1 + \ldots + h_{n-1} = t^\star$.

Using the notations

$$\begin{aligned} g_i &= h_i f(t_i, u(t_i)) - (u(t_{i+1}) - u(t_i)), \\ \psi_i &= f(t_i, e_i + u(t_i)) - f(t_i, u(t_i)) \end{aligned} \tag{2.42}$$

the estimation (2.34) can be rewritten as follows:

$$\begin{aligned} |e_{i+1}| \leq |e_i| + |g_i| + h_i|\psi_i| \leq |e_i| + |g_i| + h_i L|e_i| \leq \\ (1 + h_i L)|e_i| + |g_i| \leq \exp(h_i L)|e_i| + |g_i| \leq \exp(h_i L)\left[|e_i| + |g_i|\right]. \end{aligned} \tag{2.43}$$

Then the estimation (2.35), taking into account (2.43), results in the relation

$$\begin{aligned} |e_n| &\leq \exp(h_{n-1}L)\left[|e_{n-1}| + |g_{n-1}|\right] \\ &\leq \exp(h_{n-1}L)\left[\exp(h_{n-2}L)\left(|e_{n-2}| + |g_{n-2}|\right) + |g_{n-1}|\right] \\ &= \exp((h_{n-1} + h_{n-2})L)\left(|e_{n-2}| + |g_{n-2}| + g_{n-1}|\right) \\ &\leq \exp((h_{n-1} + h_{n-2} + \ldots + h_0)L)\left[|e_0| + \sum_{i=1}^{n} |g_{n-i}|\right]. \end{aligned} \tag{2.44}$$

Due to the relation $h_{n-1} + h_{n-2} + \ldots + h_0 = t^\star$ and the estimation

$$|g_i| \leq \frac{M_2}{2} h_i^2 \leq \frac{M_2 c^2}{2} h^2, \tag{2.45}$$

the relations (2.44) and (2.45) together imply the estimation

$$|e_n| \leq \exp(t^\star L)\left[|e_0| + hn\frac{M_2 c^2}{2}h\right] = \exp(t^\star L)\left[|e_0| + t^\star \frac{M_2 c^2}{2}h\right]. \tag{2.46}$$

The estimation (2.46) shows that on a sequence of suitably refined meshes by $h \to 0$ we have $e_n \to 0$, and moreover, $e_n = \mathcal{O}(h)$.

This proves the following statement.

Theorem 2.2.6. *The explicit Euler method is convergent, and the rate of convergence is one.*[9]

Remark 2.2.4. We can see that for the explicit Euler method the choice $y_0 = u_0$ is not necessary to obtain the convergence, i.e., the relation $\lim_{h\to 0} e_n = 0$. Obviously, it is enough to require only the relation $y_0 = u_0 + \mathcal{O}(h)$, since in this case $e_0 = \mathcal{O}(h)$. (By this choice the rate of the convergence $e_n = \mathcal{O}(h)$ still remains true.)

2.2.2 Implicit Euler method

Let us consider the θ-method by the choice $\theta = 1$. For this case the formulas (2.24) and (2.25) together generate the following numerical method:

$$y_{i+1} = y_i + h_i f(t_{i+1}, y_{i+1}), \quad i = 0, 1, \ldots, N-1, \tag{2.47}$$

where again we put $y_0 = u_0$.

Definition 2.2.7. *The one-step numerical method defined under* (2.47) *is called* implicit Euler method.

Remark 2.2.5. The Euler method of the form (2.47) is called implicit because y_{i+1}, the value of the approximation on the new time level t_{i+1}, can be obtained by solving a usually non-linear equation. E.g., for the function $f(t, u) = \sin u$, the algorithm of the implicit Euler method reads as $y_{i+1} = y_i + h_i \sin y_{i+1}$, and hence the computation of the unknown y_{i+1} requires the solution of a non-linear algebraic equation. However, when we set $f(t, u) = u$, then we have the algorithm $y_{i+1} = y_i + h_i y_{i+1}$, and from this relation y_{i+1} can be defined directly, without solving any non-linear equation.

For the error function e_i of the implicit Euler method we have the following equation:

$$\begin{aligned} e_{i+1} - e_i &= -(u(t_{i+1}) - u(t_i)) + h_i f(t_{i+1}, u(t_{i+1}) + e_{i+1}) \\ &= [h_i f(t_{i+1}, u(t_{i+1})) - (u(t_{i+1}) - u(t_i))] \\ &\quad + h_i \left[f(t_{i+1}, u(t_{i+1}) + e_{i+1}) - f(t_{i+1}, u(t_{i+1})) \right]. \end{aligned} \tag{2.48}$$

Hence, by the notations

$$\begin{aligned} g_i &= h_i f(t_{i+1}, u(t_{i+1})) - (u(t_{i+1}) - u(t_i)), \\ \psi_i &= f(t_{i+1}, u(t_{i+1}) + e_{i+1}) - f(t_{i+1}, u(t_{i+1})) \end{aligned} \tag{2.49}$$

[9] We remind that the order of the convergence is defined by the order of the global error, cf. page 17.

we arrive again at the error equation of the form (2.32).

Clearly, we have

$$u(t_{i+1}) - u(t_i) = u(t_{i+1}) - u(t_{i+1} - h) = h_i u'(t_{i+1}) - \frac{1}{2} u''(\xi_i) h_i^2, \qquad (2.50)$$

where $\xi_i \in (t_i, t_{i+1})$ is some given point. On the other side, $f(t_{i+1}, u(t_{i+1})) = u'(t_{i+1})$. Hence for g_i, defined in (2.49), we have the inequality

$$|g_i| \leq \frac{M_2}{2} h_i^2. \qquad (2.51)$$

However, for the implicit Euler method the value of ψ_i depends both on the approximation and on the exact solution at the point t_{i+1}, therefore, the proof of the convergence is more complicated than it was done for the explicit Euler method. In the following we give an elementary proof on a uniform mesh, i.e., for the case $h_i = h$.

We consider the implicit Euler method, which means that the values y_i at the mesh-points ω_h are defined by the one-step recursion (2.47). Using the Lipschitz property of the function f, from the error equation (2.32) we obtain

$$|e_{i+1}| \leq |e_i| + hL|e_{i+1}| + |g_i|, \qquad (2.52)$$

which implies the relation

$$(1 - Lh)|e_{i+1}| \leq |e_i| + |g_i|. \qquad (2.53)$$

Assume that $h < h_0 := \dfrac{1}{2L}$. Then (2.53) implies the relation

$$|e_{i+1}| \leq \frac{1}{1 - hL}|e_i| + \frac{1}{1 - hL}|g_i|. \qquad (2.54)$$

We give an estimation for $\dfrac{1}{1 - hL}$. Based on the assumption, $hL \leq 0.5$, therefore we can write this expression as

$$\begin{aligned} &1 < \frac{1}{1 - hL} = 1 + hL + (hL)^2 + \cdots + (hL)^n + \cdots = \\ &1 + hL + (hL)^2 \left(1 + hL + (hL)^2 + \dots\right) = 1 + hL + (hL)^2 \frac{1}{1 - hL}. \end{aligned} \qquad (2.55)$$

Obviously, for the values $Lh < 0.5$ the estimation $\dfrac{(hL)^2}{1 - hL} < hL$ holds. Therefore, we have the upper bound

$$\frac{1}{1 - hL} < 1 + 2hL < \exp(2hL). \qquad (2.56)$$

Since for the values $Lh \in [0, 0.5]$ we have $\dfrac{1}{1-hL} \leq 2$, therefore for the global error the substitution (2.56) into (2.54) results in the recursive relation

$$|e_i| \leq \exp(2hL)|e_{i-1}| + 2|g_i|. \tag{2.57}$$

Due to the error estimation (2.51), we have

$$|g_i| \leq \frac{M_2}{2}h^2 := g_{max} \text{ for all } i = 1, 2, \ldots, N. \tag{2.58}$$

Therefore, based on (2.57), we have the recursion

$$|e_i| \leq \exp(2hL)|e_{i-1}| + 2g_{max}. \tag{2.59}$$

The following statement is simple, and its proof by induction is left for the Reader.

Lemma 2.2.8. *Let $a > 0$ and $b \geq 0$ given numbers, and s_i ($i = 0, 1, \ldots, k-1$) such numbers that the inequalities*

$$|s_i| \leq a|s_{i-1}| + b, \quad i = 1, \ldots, k-1 \tag{2.60}$$

hold. Then the estimations

$$|s_i| \leq a^i|s_0| + \frac{a^i - 1}{a - 1}b, \quad i = 1, 2, \ldots, k \tag{2.61}$$

are valid.

Hence, we have

Corollary 2.2.9. When $a > 1$, then obviously

$$\frac{a^i - 1}{a - 1} = a^{i-1} + a^{i-2} + \cdots + 1 \leq ia^{i-1}.$$

Hence, for this case Lemma 2.2.8, instead of (2.61) yields the estimation

$$|s_i| \leq a^i|s_0| + ia^{i-1}b, \quad i = 1, 2, \ldots, k. \tag{2.62}$$

Let us apply Lemma 2.2.8 to the global error e_i. Choosing $a = \exp(2hL) > 1$, $b = 2g_{max} \geq 0$, and taking into account the relation (2.59), based on (2.62) we get

$$|e_i| \leq [\exp(2hL)]^i |e_0| + i [\exp(2hL)]^{i-1} 2g_{max}. \tag{2.63}$$

Due to the obvious relations

$$[\exp(2hL)]^i = \exp(2Lhi) = \exp(2Lt_i),$$

and

$$i\,[\exp(2hL)]^{i-1}\,2g_{max} < 2ih\exp(2Lhi)\frac{g_{max}}{h} = 2t_i\exp(2Lt_i)\frac{g_{max}}{h},$$

the relation (2.63) results in the estimation

$$|e_i| \le \exp(2Lt_i)\left[|e_0| + 2t_i\frac{g_{max}}{h}\right], \tag{2.64}$$

which holds for every $i = 1, 2, \ldots, N$.

Using the expression for g_{max} in (2.58), from the formula (2.63) we get

$$|e_i| \le \exp(2Lt_i)\left[|e_0| + M_2ht_i\right], \tag{2.65}$$

for any $i = 1, 2, \ldots, N$.

Let us apply the estimate (2.65) for the index n. (Remember that on a fixed mesh ω_h, n denotes the index for which $nh = t^\star$, where $t^\star \in [0, T]$ is some fixed point, where the convergence is analyzed.) Then we get

$$|e_n| \le \exp(2Lt_n)\left[|e_0| + M_2ht_n\right] = \exp(2Lt^\star)\left[|e_0| + M_2ht^\star\right]. \tag{2.66}$$

Since $e_0 = 0$, therefore, finally we get

$$|e_n| \le C \cdot h, \tag{2.67}$$

where $C = M_2t^\star\exp(2Lt^\star) = \mathit{constant}$. This proves the following statement.

Theorem 2.2.10. *The implicit Euler method is convergent, and the rate of convergence is one.*

Finally, we make two comments.

- The convergence on the interval $[0, t^\star]$ yields the relation

  $$\lim_{h\to 0}\max_{i=1,2,\ldots,n}|e_i| = 0.$$

  As one can easily see, the relation $|e_n| \le C \cdot h$ holds for both methods (explicit Euler method and implicit Euler method). Therefore the local truncation error $|e_n|$ can be bounded at any point uniformly on the interval $[0, t^\star]$, which means first order convergence on the interval.

- Since the implicit Euler method is implicit, in each step we must solve a – usually non-linear – equation, namely, find the root of the equation $g(s) := s - hf(t_n, s) - y_n = 0$. This can be done by using some iterative method, such as Newton's method.

2.2.3 Trapezoidal method

Let us consider the θ-method by the choice $\theta = 0.5$. For this case the formulas (2.24) and (2.25) generate the numerical method of the form

$$y_{i+1} - y_i = \frac{h_i}{2} \left[f(t_i, y_i) + f(t_{i+1}, y_{i+1}) \right], \quad i = 0, 1, \ldots, N-1, \tag{2.68}$$

where $y_0 = u_0$.

Definition 2.2.11. *The one-step method* (2.68) *is called* trapezoidal method.

We remark that the trapezoidal method is a combination of the explicit Euler method and the implicit Euler method. Clearly, this method is implicit, like the implicit Euler method. We also note that this method is sometimes also called Crank–Nicolson method, due to the application of this method to the numerical solution of the semi-discretized heat conduction equation.

By combining the error equation for the explicit Euler method and the implicit Euler method, we can easily obtain the error equation for the trapezoidal method in the form (2.32) again. One can easily verify that for this method we get

$$\begin{aligned} g_i &= \frac{1}{2} h_i \left[f(t_i, u(t_i)) + f(t_{i+1}, u(t_{i+1})) \right] - (u(t_{i+1}) - u(t_i)), \\ \psi_i &= \frac{1}{2} \left[f(t_i, u(t_i) + e_i) - f(t_i, u(t_i)) \right] + \\ &\quad \frac{1}{2} \left[f(t_{i+1}, u(t_{i+1}) + e_{i+1}) - f(t_{i+1}, u(t_{i+1})) \right]. \end{aligned} \tag{2.69}$$

Let us estimate g_i in (2.69). Using the notation $t_{i+\frac{1}{2}} = t_i + 0.5 h_i$ we develop $u(t)$ around this point into the Taylor series with the Lagrange remainder: $u(t_i) = u(t_{i+\frac{1}{2}} - h_i/2)$ and $u(t_{i+1}) = u(t_{i+\frac{1}{2}} + h_i/2)$. Then

$$u(t_{i+1}) - u(t_i) = h_i u'(t_{i+1/2}) + \frac{h_i^3}{48} (u'''(\xi_i^1) + u'''(\xi_i^2)), \tag{2.70}$$

where $\xi_i^1, \xi_i^2 \in (t_i, t_{i+1})$ are given points. On the other hand, we have

$$f(t_i, u(t_i)) + f(t_{i+1}, u(t_{i+1})) = u'(t_i) + u'(t_{i+1}). \tag{2.71}$$

Developing again the functions on the right side of (2.71) around the point $t = t_{i+\frac{1}{2}}$, we get the equality

$$\frac{1}{2} \left[f(t_i, u(t_i)) + f(t_{i+1}, u(t_{i+1})) \right] = u'(t_{i+\frac{1}{2}}) + \frac{h_i^2}{16} (u'''(\xi_i^3) + u'''(\xi_i^4)). \tag{2.72}$$

t_i	exact solution	EE	IE	TR
0.1	1.0048	1.0000	1.0091	1.0048
0.2	1.0187	1.0100	1.0264	1.0186
0.3	1.0408	1.0290	1.0513	1.0406
0.4	1.0703	1.0561	1.0830	1.0701
0.5	1.1065	1.0905	1.1209	1.1063
0.6	1.1488	1.1314	1.1645	1.1485
0.7	1.1966	1.1783	1.2132	1.1963
0.8	1.2493	1.2305	1.2665	1.2490
0.9	1.3066	1.2874	1.3241	1.3063
1.0	1.3679	1.3487	1.3855	1.3676

Table 2.3: Comparison in the maximum norm for the explicit Euler method (EE), the implicit Euler method (IE), and the trapezoidal method (TR) on the mesh with mesh-size $h = 0.1$.

Hence, using the relations (2.70) and (2.72), for the local approximation error g_i we get

$$|g_i| \leq \frac{M_3}{6} h_i^3, \quad M_3 = \max_{[0,t^\star]} |u'''(t)|. \tag{2.73}$$

We consider the test equation given in Exercise 2.13. In Table (2.3) we give the numerical solution by the above listed three numerical methods (explicit Euler method, implicit Euler method, trapezoidal method). In Table (2.4) we compare the numerical results obtained on refining meshes in the maximum norm. These results show that on some fixed mesh the explicit Euler method and the implicit Euler method give approximately the same accuracy, while the trapezoidal method is more accurate. On the refining meshes we can observe that the error function of the trapezoidal method has the order $\mathcal{O}(h^2)$, while for the explicit Euler method and the implicit Euler method the error function is of $\mathcal{O}(h)$. (These results completely correspond to the theory.)

2.2.4 Alternative introduction of the one-step methods

The discussed one-step numerical methods can be introduced in other ways, too. One possibility is the following. Let $u(t)$ be the solution of the Cauchy problem (1.6), which means that the identity (2.3) holds on the interval $[0, T]$. Integrating both sides of this identity between two neighboring mesh-points of

step-size (h)	EE	IE	TR
0.1	$1.92e-02$	$1.92e-02$	$3.06e-04$
0.01	$1.84e-03$	$1.84e-03$	$3.06e-06$
0.001	$1.84e-04$	$1.84e-04$	$3.06e-08$
0.0001	$1.84e-05$	$1.84e-05$	$3.06e-10$
0.0001	$1.84e-06$	$1.84e-06$	$5.54e-12$

Table 2.4: The error in the maximum norm for the explicit Euler method (EE), the implicit Euler method (IE), and the trapezoidal method (TR) on the mesh with mesh-size h.

the mesh ω_h, we get

$$u(t_{i+1}) - u(t_i) = \int_{t_i}^{t_{i+1}} f(t, u(t))dt, \quad t \in [0, T] \tag{2.74}$$

for any $i = 0, 1, \ldots, N-1$. To define some numerical method, on the interval $[t_i, t_{i+1}]$ we integrate numerically the right side of this relation. The simplest integration formulas are using the end-points of the interval, only. The different integration formulas result in the above methods.

- The simplest way is using the value taken on the left-hand side of the interval, i.e., the integration rule

$$\int_{t_i}^{t_{i+1}} f(t, u(t))dt \approx h_i f(t_i, u(t_i)). \tag{2.75}$$

  Then (2.74) and (2.75) together result in the formula (2.26), which is the explicit Euler method.

- Another possibility is using the value taken on the right-hand side of the interval, i.e., the integration rule

$$\int_{t_i}^{t_{i+1}} f(t, u(t))dt \approx h_i f(t_{i+1}, u(t_{i+1})). \tag{2.76}$$

  Then (2.74) and (2.76) result in the formula (2.47), which is the implicit Euler method.

- When we use the trapezoidal rule for the numerical integration in (2.74), i.e., the formula

$$\int_{t_i}^{t_{i+1}} f(t, u(t))dt \approx \frac{h_i}{2} \left[f(t_i, u(t_i)) + f(t_{i+1}, u(t_{i+1})) \right] \tag{2.77}$$

is applied, then (2.74) and (2.77) together result in the formula (2.68), i.e., the trapezoidal method.

- Using the numerical integration formula

$$\int_{t_i}^{t_{i+1}} f(t,u(t))dt \approx h_i \left[(1-\theta)f(t_i,u(t_i)) + \theta f(t_{i+1},u(t_{i+1}))\right], \quad (2.78)$$

we obtain the θ-method, which was already defined by formulas (2.24)-(2.25).

In order to re-obtain the numerical formulas, another approach is the following. We consider the identity (2.3) at some fixed mesh-point, and for the left side we use some numerical derivation rule.

- When we use the point $t = t_i$, and we write the identity (2.3) at this point, then we get $u'(t_i) = f(t_i, u(t_i))$. Let us apply the forward finite difference approximation to the derivative on the left-hand side, then we get the approximation $u'(t_i) \simeq (u(t_{i+1}) - u(t_i))/h_i$. These two formulas together generate the explicit Euler method.

- When we write the identity (2.3) at the point $t = t_{i+1}$, and we use the retrograde numerical finite difference derivation, then we obtain the implicit Euler method.

- The arithmetic mean of the above formulas result in the trapezoidal method.

The behavior of the above methods for the equation $u' = 1 - t\sqrt[3]{u}$ can be seen on the animation under the link
`http://math.fullerton.edu/mathews/a2001/Animations/Animations9.html`

2.3 The basic concepts of general one-step methods

In this section on the uniform mesh

$$\omega_h := \{t_i = ih;\ i = 0, 1, \ldots, n;\ h = T/N\}$$

(or on their sequence) we formulate the general form of one-steps methods, and we define some basic concepts of such methods. We consider the numerical method of the form

$$y_{i+1} = y_i + h\Phi(h, t_i, y_i, y_{i+1}), \quad (2.79)$$

where Φ is some given function. In the sequel, the numerical method, defined by (2.79) is called Φ *numerical method* (in short, Φ-*method*).

Remark 2.3.1. By some special choice of the function Φ the previously discussed one-step methods can be written in the form (2.79). E.g.

- by the choice $\Phi(h, t_i, y_i, y_{i+1}) = f(t_i, y_i)$ we get the explicit Euler method;
- by the choice $\Phi(h, t_i, y_i, y_{i+1}) = f(t_i + h, y_{i+1})$ we get the implicit Euler method;
- by the choice $\Phi(h, t_i, y_i, y_{i+1}) = 0.5\,[f(t_i, y_i) + f(t_i + h, y_{i+1})]$ we get the trapezoidal method,
- by the choice $\Phi(h, t_i, y_i, y_{i+1}) = (1 - \theta)f(t_i, y_i) + \theta f(t_i + h, y_{i+1})$ we get the θ-method.

Definition 2.3.1. *The numerical methods with $\Phi = \Phi(h, t_i, y_i)$ (i.e., the function Φ is independent of y_{i+1}) are called* explicit methods. *When Φ really depends on y_{i+1}, i.e., $\Phi = \Phi(h, t_i, y_i, y_{i+1})$, the numerical method is called* implicit.

As before, $u(t)$ denotes the exact solution of the Cauchy problem (2.1)–(2.2), and let $t_\star \in (0, T]$ be some given point.

Definition 2.3.2. *The function $e_{t_\star}(h) = y_n - u(t_\star), (nh = t_\star)$ is called* global discretization error *of the Φ numerical method at the point $t_\star$. We say that the Φ numerical method is* convergent *at the point $t_\star$, when the relation*

$$\lim_{h \to 0} e_h(t_\star) = 0 \tag{2.80}$$

holds. When the Φ numerical method is convergent at each point of the interval $[0, T]$, then the numerical method is called convergent method. *The order of the convergence to zero in* (2.80) *is called* order of the convergence *of the Φ-method.*

The local behavior of the Φ-method can be characterized by the value which is obtained after one step using the exact solution as starting value. This value $\hat{y}_{i+1}$, being an approximation to the exact solution, is defined as

$$\hat{y}_{i+1} = u(t_i) + h\Phi(h, t_i, u(t_i), \hat{y}_{i+1}). \tag{2.81}$$

Definition 2.3.3. *The mesh-function* $l_i(h) = \hat{y}_i - u(t_i)$ *(where* $t_i = ih \in \omega_h$*) is called* local discretization error *of the* Φ*-method, defined in* (2.79)*, at the point* t_i*.*

For the implicit Φ-methods the calculation of the local discretization error is usually difficult, because one has to know the value $\hat{y}_{i+1}$ from the formula (2.81). To avoid this difficulty, we introduce the mesh-function

$$g_i(h) = -u(t_{i+1}) + u(t_i) + h\Phi(h, t_i, u(t_i), u(t_{i+1})). \tag{2.82}$$

The order of this expression (using the equation in the Cauchy problem) can be defined directly, without knowing the exact solution, by developing into Taylor series.

Definition 2.3.4. *The function* $g_i(h)$ *is called* local truncation error *of the* Φ*-method, defined in* (2.79)*, at the point* t_i*. We say that the* Φ*-method is* consistent of order r*, when*

$$g_i(h) = \mathcal{O}(h^{r+1}) \tag{2.83}$$

with some constant $r > 0$ *for any mesh-point* $t_i \in \omega_h$*.*

Obviously, the truncation error and its order shows how exactly the exact solution satisfies the formula of the Φ-method.[10]

Remark 2.3.2. Based on the estimations (2.38), (2.51) and (2.73) we can see that both the explicit Euler method and the implicit Euler method are consistent of first order, while the trapezoidal method has of second order consistency. An elementary calculation shows that the θ-method is of second order only for $\theta = 0.5$ (which means the trapezoidal method), otherwise it is consistent of first order .

Further we assume that the function Φ in the numerical method (2.79) satisfies the Lipschitz condition w.r.t. its third and fourth variables, which means the following: there exist constants $L_3 \geq 0$ and $L_4 \geq 0$ such that for any numbers s_1, s_2, p_1 and p_2 the inequality

$$|\Phi(h, t_i, s_1, p_1) - \Phi(h, t_i, s_2, p_2)| \leq L_3|s_1 - s_2| + L_4|p_1 - p_2| \tag{2.84}$$

holds for all $t_i \in \omega_h$ and $h > 0$. When the function Φ is independent on y_i (or on y_{i+1}), then $L_3 = 0$ (or $L_4 = 0$).

Remark 2.3.3. According to the Remark 2.3.1, when the function f satisfies the Lipshitz condition w.r.t. its second variable, then Φ, corresponding to the θ-method for any θ satisfies (2.84) with some suitably chosen L_3 and L_4.

[10]The global discretization error, defined as $e_{t_\star}(h) = y_n - u(t_\star)$ is also called *global truncation error*, which is, in fact the accumulation of the local truncation error over all of the iterations, assuming perfect knowledge of the true solution at the initial time step.

2.3.1 Convergence of the one-step methods

Further we analyze the convergence of the Φ-method, given by (2.84). We assume that it satisfies the Lipschitz condition w.r.t. its second and third variables, and it is consistent of order r. For the sake of simplicity, we introduce the notations

$$e_{t_i}(h) = e_i, \quad g_i(h) = g_i, \quad l_i(h) = l_i. \tag{2.85}$$

According to the definition of the local and global discretization errors, we have the following relations:

$$e_{i+1} = y_{i+1} - u(t_{i+1}) = (y_{i+1} - \hat{y}_{i+1}) + (\hat{y}_{i+1} - u(t_{i+1})) = (y_{i+1} - \hat{y}_{i+1}) + l_i \tag{2.86}$$

Hence,

$$|e_{i+1}| \le |y_{i+1} - \hat{y}_{i+1}| + |l_i| \tag{2.87}$$

and we give an overestimate for both expressions on the right side of (2.87).

For the local discretization error we have

$$\begin{aligned} l_{i+1} &= \hat{y}_{i+1} - u(t_{i+1}) \\ &= u(t_i) + h\Phi(h, t_i, u(t_i), \hat{y}_{i+1})) - u(t_{i+1}) \\ &= -u(t_{i+1}) + u(t_i) + h\Phi(h, t_i, u(t_i), u(t_{i+1})) \\ &+ h\left[\Phi(h, t_i, u(t_i), \hat{y}_{i+1})) - \Phi(h, t_i, u(t_i), u(t_{i+1}))\right] \\ &= g_i + h\left[\Phi(h, t_i, u(t_i), \hat{y}_{i+1})) - \Phi(h, t_i, u(t_i), u(t_{i+1}))\right]. \end{aligned} \tag{2.88}$$

Hence, using the condition (2.84), we get

$$|l_{i+1}| \le |g_i| + hL_4|\hat{y}_{i+1} - u(t_{i+1})| = |g_i| + hL_4|l_{i+1}|. \tag{2.89}$$

Therefore, based on (2.89), the estimation

$$|l_{i+1}| \le \frac{1}{1 - hL_4}|g_i| \tag{2.90}$$

holds.

Remark 2.3.4. The inequality (2.90) shows that the order of the local approximation error cannot be less than the order of the consistency. (I.e., if the method is consistent of order r, then he local approximation error tends to zero with a rate not less than r.)

Let us estimate the first term on the right side of (2.87).

We have

$$\begin{aligned}|y_{i+1}-\hat{y}_{i+1}| = \\ &|(y_i+h\Phi(h,t_i,y_i,y_{i+1}))-(u(t_i)+h\Phi(h,t_i,u(t_i),\hat{y}_{i+1}))| \\ &\leq |e_i|+h|\Phi(h,t_i,y_i,y_{i+1})-\Phi(h,t_i,u(t_i),\hat{y}_{i+1})| \\ &\leq |e_i|+hL_3|y_i-u(t_i)|+hL_4|y_{i+1}-\hat{y}_{i+1}| \\ &= (1+hL_3)|e_i|+hL_4|y_{i+1}-\hat{y}_{i+1}|.\end{aligned} \tag{2.91}$$

Hence, based on (2.91), the inequality

$$|y_{i+1}-\hat{y}_{i+1}| \leq \frac{1+hL_3}{1-hL_4}|e_i| \tag{2.92}$$

holds. Using the relations (2.90) and (2.92), the inequality (2.87) can be rewritten as

$$|e_{i+1}| \leq \frac{1+hL_3}{1-hL_4}|e_i| + \frac{1}{1-hL_4}|g_i|. \tag{2.93}$$

Let us introduce the notations

$$\mu = \mu(h) = \frac{1+hL_3}{1-hL_4}, \quad \chi = \chi(h) = \frac{1}{1-hL_4}. \tag{2.94}$$

Remark 2.3.5. With the notations (2.94) we have

$$\mu = 1 + h\frac{L_3+L_4}{1-hL_4}, \tag{2.95}$$

therefore $\mu = 1+\mathcal{O}(h)$. Consequently, we can choose constants h_0, μ_0 and λ_0 such that the estimations

$$\mu = \mu(h) \leq 1+\mu_0 h, \quad \chi = \chi(h) \leq \chi_0, \quad \forall h \in (0,h_0) \tag{2.96}$$

are true[11].

Using the notation (2.94), the estimation (2.93) can be written as

$$|e_{i+1}| \leq \mu|e_i| + \chi|g_i|. \tag{2.97}$$

Applying the relation (2.97) recursively, we obtain the inequality

$$\begin{aligned}|e_n| &\leq \mu|e_{n-1}|+\chi|g_{n-1}| \leq \mu\left[\mu|e_{n-2}|+\chi|g_{n-2}|\right]+\chi|g_{n-1}| = \mu^2|e_{n-2}| \\ &+\chi\left[\mu|g_{n-2}|+|g_{n-1}|\right] \leq \ldots \leq \mu^n|e_0|+\chi\sum_{i=0}^{n-1}\mu^i|g_{n-1-i}| \\ &\leq \mu^n\left[|e_0|+\chi\sum_{i=0}^{n-1}|g_{n-1-i}|\right].\end{aligned} \tag{2.98}$$

[11] E.g., $h_0 = \frac{1}{2L_4}, \mu_0 = 2(L_3+L_4)$ and $\chi_0 = 2$.

Using (2.96), for any $h \in (0, h_0)$ we have $\chi \leq \chi_0$ and

$$\mu^n \leq (1 + \mu_0 h)^n \leq \exp(\mu_0 h n) = \exp(\mu_0 t_\star). \tag{2.99}$$

Therefore, by using (2.98), the estimate

$$|e_n| \leq \exp(\mu_0 t_\star) \left[|e_0| + \chi_0 \sum_{i=0}^{n-1} |g_{n-1-i}| \right] \tag{2.100}$$

holds. Since the Φ-method is consistent of order r, therefore, due to the definition (2.80), for a sufficiently small h we have $|g_i| \leq c_0 h^{r+1}$ with some constant $c_0 \geq 0$. Hence, for small values of h we get

$$\sum_{i=0}^{n-1} |g_{n-1-i}| \leq n c_0 h^{r+1} = c_0 t_\star h^r. \tag{2.101}$$

Comparing the formulas (2.100) and (2.101), we get the estimation

$$|e_n| \leq \exp(\mu_0 t_\star) \left[|e_0| + c_1 h^r \right], \tag{2.102}$$

where $c_1 = \chi_0 c_0 t_\star$=constant.

Since $e_0 = 0$, therefore from (2.102) the following statement is proven.

Theorem 2.3.5. *Assume that the Φ numerical method, defined by the formula* (2.79)*, has the following properties:*

- *it is consistent of order r;*
- *for the function Φ the Lipschitz condition* (2.84) *is satisfied.*

Then the Φ-method is convergent on the interval $[0, T]$ and the order of the convergence is r.

Corollary 2.3.6. Using Remark 2.3.2 and Remark 2.3.3, it is clear that the θ-method for $\theta = 0.5$ is convergent in second order, otherwise its order of convergence is equal to one.

Remark 2.3.6. In this part the convergence was proven from the relation (2.97), namely, using two properties of the terms appearing in this expression:

- the method is consistent, i.e., $g_i = \mathcal{O}(h^{r+1})$ with some positive number r, and the function $\chi(h)$ is bounded;

- for the function $\mu(h)$ the estimation (2.96) holds. (This property shows that passing from some time level to the next one, with increasing the number of the steps (i.e., with decreasing h) the error remains bounded. This property is called *stability of the method*.

Hence, Theorem 2.3.5 shows that for well-posed Cauchy problems the consistency and stability of the Φ-method guarantees the convergence. (The stability can be guaranteed by the Lipschitz condition for the function f on right side of the differential equation in the Cauchy problem (2.1)-(2.2).)

2.4 Runge–Kutta method

As we have seen, the maximum accuracy of the investigated one-step methods (explicit Euler method, implicit Euler method, trapezoidal method, θ-method) for the Cauchy problem (2.1)-(2.2) is two. However, from a practical point of view, this accuracy is not enough: typically we require the construction of numerical methods with higher order of accuracy. The accuracy of the Taylor method is higher, but, in this case the realization of this method requires rather complicated preliminary analysis. (Determination of all partial derivatives and its evaluation at given points.) In this section we show that, by use of some simple idea, the task of determining the partial derivatives can be omitted.

2.4.1 Second order Runge–Kutta methods

Let us consider again the Cauchy problem (2.1)–(2.2). In order to introduce the Runge–Kutta methods, first of all we define a one-step method of second order accuracy, which is different from the trapezoidal method.
Let us define the first terms of the Taylor series of the function $u(t)$ in the form (2.6) at the point $t = t^\star + h$. Then

$$u(t^\star + h) = u(t^\star) + hu'(t^\star) + \frac{h^2}{2!}u''(t^\star) + \mathcal{O}(h^3). \tag{2.103}$$

Using the derivatives (2.4), and introducing the notations

$$f = f(t^\star, u(t^\star)), \quad \partial_i f = \partial_i f(t^\star, u(t^\star)), \quad \partial_{ij} f = \partial_{ij} f(t^\star, u(t^\star)), \quad \text{etc.},$$

the equation (2.103) can be rewritten as

$$\begin{aligned} u(t^\star + h) =& u(t^\star) + hf + \frac{h^2}{2!}(\partial_1 f + f\partial_2 f) + \mathcal{O}(h^3) \\ =& u(t^\star) + \frac{h}{2}f + \frac{h}{2}[f + h\partial_1 f + hf\partial_2 f] + \mathcal{O}(h^3). \end{aligned} \tag{2.104}$$

Since [12]

$$f(t^\star + h, u(t^\star) + hf(t^\star, u(t^\star)) = f + h\partial_1 f + hf\partial_2 f + \mathcal{O}(h^2), \tag{2.105}$$

therefore (2.104) can be written in the form

$$u(t^\star + h) = u(t^\star) + \frac{h}{2}f + \frac{h}{2}(f(t^\star + h, u(t^\star) + hf(t^\star, u(t^\star))) + \mathcal{O}(h^3). \tag{2.106}$$

Therefore, applying the formula (2.106) at some arbitrary mesh-point $t_i = t^\star$ of ω_h, we can define the following one-step, explicit numerical method:

$$y_{i+1} = y_i + \frac{h}{2}f(t_i, y_i) + \frac{h}{2}f(t_{i+1}, y_i + hf(t_i, y_i)). \tag{2.107}$$

Let us introduce the notations

$$k_1 = f(t_i, y_i); \quad k_2 = f(t_{i+1}, y_i + hf(t_i, y_i)) = f(t_i + h, y_i + hk_1). \tag{2.108}$$

Then the method (2.107) can be written in the form

$$y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2). \tag{2.109}$$

Definition 2.4.1. *The one-step, explicit numerical method* (2.108)-(2.109) *is called* Heun method[13]*.*

Remark 2.4.1. Based on (2.106), we have

$$u(t^\star + h) - u(t^\star) - \frac{h}{2}f - \frac{h}{2}(f(t^\star + h, u(t^\star) + hf(t^\star, u(t^\star))) = \mathcal{O}(h^3). \tag{2.110}$$

This means that the exact solution of the Cauchy problem (2.1)-(2.2) satisfies the formula of the Heun method (2.107) with the accuracy $\mathcal{O}(h^3)$, which means that the Heun method is of second order.

Some further details of the Heun method can be found under the link
`http://math.fullerton.edu/mathews/n2003/Heun'sMethodMod.html`
Here one can also see an animation for the behavior of the numerical solution, obtained by the Heun method, for the differential equation $u' = 1 - t\sqrt[3]{u}$.

[12] We recall that the first order Taylor polynomial of the function $f : Q_T \to \mathbb{R}$ around the point (t, u) , for arbitrary constants $c_1, c_2 \in \mathbb{R}$ can be written as $f(t + c_1h, u + c_2h) = f(t, u) + c_1h\partial_1 f(t, u) + c_2h\partial_2 f(t, u) + \mathcal{O}(h^2)$.

[13] The method is named after Karl L. W. M. Heun (1859–1929), who was a German mathematician. He introduced Heun's equation, Heun functions, and Heun's method.

Let us define other one-step methods of second order accuracy. Obviously, the parameterized form of the Heun method (2.106) is the following:

$$u(t^\star+h) = u(t^\star)+\sigma_1 hf(t^\star, u(t^\star))+\sigma_2 hf(t^\star+a_2h, u(t^\star)+b_{21}hf(t^\star, u(t^\star)))+\mathcal{O}(h^3), \tag{2.111}$$

where σ_1, σ_2, a_2 and b_{21} are some free parameters. Let us write the equation (2.111) at the point $t = t_i$. This generates the one-step numerical method of the form

$$y_{i+1} = y_i + \sigma_1 hf(t_i, y_i) + \sigma_2 hf(t_i + a_2h, y_i + b_{21}hf(t_i, y_i)). \tag{2.112}$$

Remark 2.4.2. It is subservient to write the parameters of the method (2.112) in the following form:

$$\begin{array}{c|cc} 0 & & \\ a_2 & b_{21} & \\ \hline & \sigma_1 & \sigma_2 \end{array} \tag{2.113}$$

For more explanation we refer to the Definition 2.4.6.

By developing the right side of (2.111) into Taylor series , we obtain the equality

$$\begin{aligned} u(t^\star + h) =& u(t^\star) + \sigma_1 hf + \sigma_2 h[f + a_2h\partial_1 f + b_{21}hf\partial_2 f] + \mathcal{O}(h^3) \\ =& u(t^\star) + (\sigma_1 + \sigma_2)hf + h^2[a_2\sigma_2\partial_1 f + \sigma_2 b_{21} f\partial_2 f] + \mathcal{O}(h^3). \end{aligned} \tag{2.114}$$

Let us use Remark 2.4.1, with comparing the formulas (2.104) and (2.114). Then we conclude that the numerical method, defined by (2.112) has second order accuracy if and only if

$$\begin{aligned} \sigma_1 + \sigma_2 &= 1 \\ a_2\sigma_2 &= 0.5 \\ b_{21}\sigma_2 &= 0.5. \end{aligned} \tag{2.115}$$

Rewriting the formulas (2.112), our results can be summarized as follows.

Theorem 2.4.2. *Let the parameters σ_1, σ_2, a_2 and b_{21} be solution of the equation* (2.115)*. Then the explicit, one-step method*

$$k_1 = f(t_i, y_i), \quad k_2 = f(t_i + a_2h, y_i + hb_{21}k_1), \tag{2.116}$$

$$y_{i+1} = y_i + h(\sigma_1 k_1 + \sigma_2 k_2) \tag{2.117}$$

is of second order accuracy.

Definition 2.4.3. *The numerical method* (2.116)-(2.117) *under the condition* (2.115) *is called* second order Runge–Kutta method *and it is denoted by RK2.*

Let us examine the system of algebraic equations (2.115), which defines the RK2 method. For the four unknowns we have only three equations, therefore the solution is not unique. One can easily see that for any free parameter $\sigma \neq 0$ the solution of the system is

$$\sigma_2 = \sigma, \quad \sigma_1 = 1 - \sigma, \quad a_2 = b_{21} = 1/2\sigma. \tag{2.118}$$

Hence, the RK2 methods form a one-parameter family, and the parameters of the method have the form (using the representation (2.113))

$$\begin{array}{c|cc} 0 & & \\ 1/2\sigma & 1/2\sigma & \\ \hline & 1-\sigma & \sigma \end{array} \tag{2.119}$$

Remark 2.4.3. The RK2 method, corresponding to the choice $\sigma = 0.5$ is the Heun method. The choice $\sigma = 1$ is interesting. For this case $\sigma_1 = 0$, $\sigma_2 = 1$ and $a_2 = b_{21} = 0.5$, and these parameters imply the following method:

$$k_1 = f(t_i, y_i), \quad k_2 = f(t_i + 0.5h, y_i + 0.5hk_1), \quad y_{i+1} = y_i + hk_2. \tag{2.120}$$

Definition 2.4.4. *The second order numerical method of the form* (2.120) *is called* modified Euler method.[14] *3*

We note that the Heun method and the modified Euler method can be also introduced by some modification of the simplest Euler methods, investigated before. Namely,

- when in the formula of the trapezoidal method (2.68)

$$y_{i+1} - y_i = \frac{h_i}{2}\left[f(t_i, y_i) + f(t_{i+1}, y_{i+1})\right]$$

  in the implicit part $f(t_{i+1}, y_{i+1})$ we replace y_{i+1} by its approximation, obtained by the explicit Euler method, $\tilde{y}_{i+1} = y_i + hf(t_i, y_i)$, then we get the Heun method.

[14] Alternatively, this method is also called *improved Euler method.*

- For the modified Euler method, at the midpoint $t = t_i + 0.5h = t_{i+0.5}$ of the interval $[t_i, t_{i+1}]$ we define an approximation to the exact solution $u(t)$ by the explicit Euler method, i.e., we define the value $\tilde{y}_{i+0.5} = y_i + 0.5hf(t_i, y_i)$. Then, using this value for the approximation on the interval $[t_i, t_{i+1}]$ we take another explicit Euler step on the whole interval.

The value of the parameters of the above second order methods in the form of table (2.119) are the following. For the Heun method it has the form

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 0.5 & 0.5 \end{array} \tag{2.121}$$

while for the modified Euler method we have

$$\begin{array}{c|cc} 0 & & \\ 0.5 & 0.5 & \\ \hline & 0 & 1 \end{array} \tag{2.122}$$

Remark 2.4.4. Is there a special choice for the arbitrary parameter σ such that the RK2 method is not only of second order but of third order? The answer to this question is negative, which can be seen from the following example. In the Cauchy problem (2.1)-(2.2) we put $f(t, u) = u$. Hence, for the solution the relation $u'(t) = u(t)$ holds. Differentiating this equality, we obtain $u''(t) = u'(t)$. Hence $u''(t) = u'(t) = u(t)$. On the other hand, due to the choice of function f, we have $f(t_i, y_i) = y_i$. Therefore, the method (2.112) applied to this problem yields the following:

$$\begin{aligned} y_{i+1} &= y_i + \sigma_1 h y_i + \sigma_2 h(y_i + b_{21} h f(t_i, y_i)) = y_i + \sigma_1 h y_i + \sigma_2 h(y_i + b_{21} h y_i) \\ &= y_i + h y_i[\sigma_1 + \sigma_2 + h\sigma_2 b_{21}] = y_i[1 + (\sigma_1 + \sigma_2)h + \sigma_2 b_{21} h^2]. \end{aligned} \tag{2.123}$$

Substituting the values (2.118) (which are necessary for the second order accuracy), the RK2 method for this problem reads as

$$y_{i+1} = y_i(1 + h + \frac{h^2}{2}). \tag{2.124}$$

We can observe that this method is independent of the parameter σ. We substitute the exact solution $u(t)$ into the formula (2.124), i.e., we define the local approximation error. Then

$$g_i = u(t_{i+1}) - u(t_i)(1 + h + \frac{h^2}{2}). \tag{2.125}$$

When we develop $u(t_{i+1})$ into Taylor series around the point $t = t_i$, due to the relations $u''(t_i) = u'(t_i) = u(t_i)$, it has the form $u(t_{i+1}) = u(t_i)(1 + h + h^2/2) + \mathcal{O}(h^3)$. This means that $g_i = \mathcal{O}(h^3)$ for any choice of the parameter σ, i.e., the RK2 method is of second order in this problem.

2.4.2 Higher order Runge–Kutta methods

For real-life problems first and second order methods are not applicable, because within reasonable computational time we cannot achieve the numerical solution with some prescribed and required (by the application) accuracy. Our aim is to construct *higher order accurate methods* on the basis of formula (2.112).

We re-consider the numerical method written in the form (2.116)-(2.117). As we have shown, this method is exact at most in second order. (The method has second order accuracy when the condition (2.115) is satisfied for its parameters.) Therefore, to get higher order accurate numerical methods, we have to introduce new parameters and give some conditions for their choice. Based on the formulas (2.116)-(2.117), the following generalization seems to be natural:

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + a_2 h, y_i + h b_{21} k_1), \\ k_3 &= f(t_i + a_3 h, y_i + h b_{31} k_1 + h b_{32} k_2), \end{aligned} \tag{2.126}$$

and the approximation on the new mesh-point is defined as

$$y_{i+1} = y_i + h(\sigma_1 k_1 + \sigma_2 k_2 + \sigma_3 k_3). \tag{2.127}$$

The parameters of this method, according to the table (2.113), can be written as follows

$$\begin{array}{c|ccc} 0 & & & \\ a_2 & b_{21} & & \\ a_3 & b_{31} & b_{32} & \\ \hline & \sigma_1 & \sigma_2 & \sigma_3 \end{array} \tag{2.128}$$

Our aim is to define the parameters in (2.126) in such a way that the corresponding numerical method is of third order accuracy. To get this condition, we have to define again the local approximation error, as it was done before. After some long (but not difficult) calculation we obtain the following result.

Theorem 2.4.5. *The numerical method* (2.126) *has third order accuracy if and only if the conditions*

$$\begin{aligned} a_2 = b_{21}, \quad & a_3 = b_{31} + b_{32}, \\ a_3(a_3 - a_2) - b_{32} a_2 (2 - 3a_2) = 0, \quad & \sigma_3 b_{32} a_2 = 1/6, \\ \sigma_2 a_2 + \sigma_3 a_3 = 1/2, \quad & \sigma_1 + \sigma_2 + \sigma_3 = 1 \end{aligned} \tag{2.129}$$

are satisfied.

Clearly, (2.129) yields six equations (conditions) for eight unknown values. From the possible solution we give two cases.

- The method with the parameters

$$\begin{array}{c|ccc} 0 & & & \\ 1/3 & 1/3 & & \\ 2/3 & 0 & 2/3 & \\ \hline & 1/4 & 0 & 3/4 \end{array} \tag{2.130}$$

  is very popular in the different applications.

- The third order method

$$\begin{array}{c|ccc} 0 & & & \\ 1/2 & 1/2 & & \\ 1 & -1 & 2 & \\ \hline & 1/6 & 2/3 & 1/6 \end{array} \tag{2.131}$$

  is also often used. We note that this method has accuracy $\mathcal{O}(h^5)$ for problems with $f(t,u) = f(t)$ with the Simpson formula. Therefore, this method is recommended also for problems where the partial derivative $\partial_2 f$ is close to zero.

When we want a higher than third order method ($p > 3$), then we need some more generalization. This can be done in the following form, which is the natural generalization of the methods (2.116)-(2.117) and (2.126).

Let $m \geq 1$ be some given integer. We define the following, so called *m-stage explicit Runge–Kutta method*:

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + a_2 h, y_i + h b_{21} k_1), \\ k_3 &= f(t_i + a_3 h, y_i + h b_{31} k_1 + h b_{32} k_2), \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ k_m &= f(t_i + a_m h, y_i + h b_{m1} k_1 + h b_{m2} k_2 + \ldots + h b_{m,m-1} k_{m-1}) \end{aligned} \tag{2.132}$$

$$y_{i+1} = y_i + h(\sigma_1 k_1 + \sigma_2 k_2 + \ldots + \sigma_m k_m). \tag{2.133}$$

By specifying the values of the parameters in this formula, we define the concrete numerical method. As before, we can write the parameters in a table:

$$\begin{array}{c|cccc} 0 & & & & \\ a_2 & b_{21} & & & \\ a_3 & b_{31} & b_{32} & & \\ \dots & \dots & \dots & \dots & \\ a_m & b_{m1} & b_{m2} & \dots & b_{m,m-1} \\ \hline & \sigma_1 & \sigma_2 & \dots & \sigma_m \end{array} \tag{2.134}$$

To write the method in a compact form, we introduce some notations. Let the symbols $\boldsymbol{\sigma}, \mathbf{a} \in \mathbb{R}^m$ denote the row-vectors with coordinates σ_i and a_i, respectively, (where we always assume that $a_1 = 0$). Let $\mathbf{B} \in R^{m \times m}$ denote the matrix with the elements b_{ij}, which is a strictly lower triangular matrix, i.e.,

$$\mathbf{B}_{ij} = \begin{cases} b_{ij}, & \text{for } i > j, \\ 0, & \text{for } i \leq j. \end{cases}$$

Definition 2.4.6. *For some explicit Runge–Kutta method the corresponding table of the parameters in the form*

$$\begin{array}{c|c} \boldsymbol{a}^\top & \boldsymbol{B} \\ \hline & \boldsymbol{\sigma} \end{array} \tag{2.135}$$

is called Butcher tableau.

In the sequel, when we specify some explicit Runge–Kutta method, then we list the lower triangular part of the matrix $\mathbf{B}$ only.

For some fixed explicit Runge–Kutta method the order of the consistency can be defined by some simple, however usually cumbersome computation. This can be done in the following steps.

- First on the right side of the formulas (2.132)-(2.133) we replace y_i by the values $u(t_i)$.
- Then on the left side of this formula we replace y_{i+1} by $u(t_i + h)$.
- We compute the difference between the two sides, and we define its order in h.

Executing these steps for an explicit Runge–Kutta method, we get the condition by which the given explicit Runge–Kutta method is of p-th order.

Remark 2.4.5. Before giving the condition of consistency for a general explicit Runge–Kutta method, we note the following. In the condition of the second order (2.115) and the third order (2.129), we can observe that any i-th row-sum of the matrix $\mathbf{B}$ is equal to the value a_i. Because this is true for any consistent method, this implies that in the Butcher tableau only the matrix $\mathbf{B}$ and the vector $\boldsymbol{\sigma}$ can be chosen arbitrarily, and the vector $\mathbf{a}$ is defined from the relation $\mathbf{a} = \mathbf{Be}$, where $\mathbf{e} = [1, 1, \ldots, 1] \in \mathbb{R}^m$.

Let us write the condition of consistency of explicit Runge–Kutta methods in a compact way. We will use the notations

$$\mathbf{a}^n = [a_1^n, a_2^n, \ldots, a_m^n,] \in \mathbb{R}^m, \quad \mathbf{A} = diag[a_1, a_2, \ldots, a_m] \in \mathbb{R}^{m\times m}.$$

Then the condition of p-th order consistency of an explicit Runge–Kutta method for $p = 1, 2, 3, 4$ imposes the following requirements for the elements of the matrix $\mathbf{B}$ and the vector $\boldsymbol{\sigma}$:[15]

order(p)	condition
1	$\boldsymbol{\sigma} \cdot \mathbf{e}^\top = 1$
2	$\boldsymbol{\sigma} \cdot \mathbf{a}^\top = 1/2$
3	$\boldsymbol{\sigma} \cdot (\mathbf{a}^2)^\top = 1/3 \quad \boldsymbol{\sigma} \cdot \mathbf{Ba}^\top = 1/6$
4	$\boldsymbol{\sigma} \cdot (\mathbf{a}^3)^\top = 1/4 \quad \boldsymbol{\sigma} \cdot \mathbf{ABa}^\top = 1/8$
	$\boldsymbol{\sigma} \cdot \mathbf{B}(\mathbf{a}^2)^\top = 1/12 \quad \boldsymbol{\sigma} \cdot \mathbf{B}^2\mathbf{a}^\top = 1/24$

(2.136)

Hence, the following statement is true.

Theorem 2.4.7. *The explicit Runge–Kutta method defined by the Butcher tableau* (2.135) *is consistent if and only if the conditions*

$$\boldsymbol{Be} = \boldsymbol{a}; \quad \boldsymbol{\sigma} \cdot \boldsymbol{e}^\top = 1 \tag{2.137}$$

are satisfied, i.e., it is true under the conditions

$$\sum_{k=1}^{m} b_{ik} = a_i \text{ for all } i = 1, 2, \ldots, m \quad \text{and} \quad \sum_{k=1}^{m} \sigma_k = 1. \tag{2.138}$$

The most popular explicit Runge–Kutta method is the method given in Table 2.5, which is a fourth order method.

The algorithm of this method (i.e., how y_{i+1}, the approximation at the mesh-point t_{i+1} can be defined from the known approximation y_i at the mesh-point t_i) is the following:

[15]These conditions can be obtained by some cumbersome computations see e.g. in [4].

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Table 2.5: Butcher tableau of the fourth order explicit Runge–Kutta method.

- By the formulas

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + 0.5h, y_i + 0.5hk_1) \\ k_3 &= f(t_i + 0.5h, y_i + 0.5hk_2) \\ k_4 &= f(t_i + h, y_i + hk_3) \end{aligned} \tag{2.139}$$

  we define the values k_1, k_2, k_3 and k_4.

- By the formula

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{2.140}$$

  we define the new approximation.

Remark 2.4.6. What is the connection between the number of the stages (m) and the order of consistency (p) for an explicit Runge–Kutta method? As we have seen, for the one-stage explicit Euler method the method is of first order, for the two-stage methods (Heun method, modified Euler method) the methods are of second order, for the three-stage method (2.130)-(2.131) it has third order, and for the four-stage method (2.139)-(2.140) it is consistent in fourth order. This means that for $m = 1, 2, 3, 4$ the stage and the order of consistency are equal. However, for the cases $m \geq 5$ this is not true anymore: the order of the method is less than the number of stages, i.e., $p < m$. The relation between them (up to $m = 10$) is the following:

m	$1, 2, 3, 4$	$5, 6, 7$	$8, 9, 10$
$p(m)$	m	$m-1$	$m-2$

(2.141)

We enclose an interactive animation, called *onestep_exp.exe*. The program can be downloaded from
`http://www.cs.elte.hu/~faragois/nummod_jegyzet_prog/`.

The image of this program on the screen can be seen in Figure 2.1.

Alternatively, this program suggests three initial-value problems as test problems. The chosen numerical methods can be the explicit Euler method,

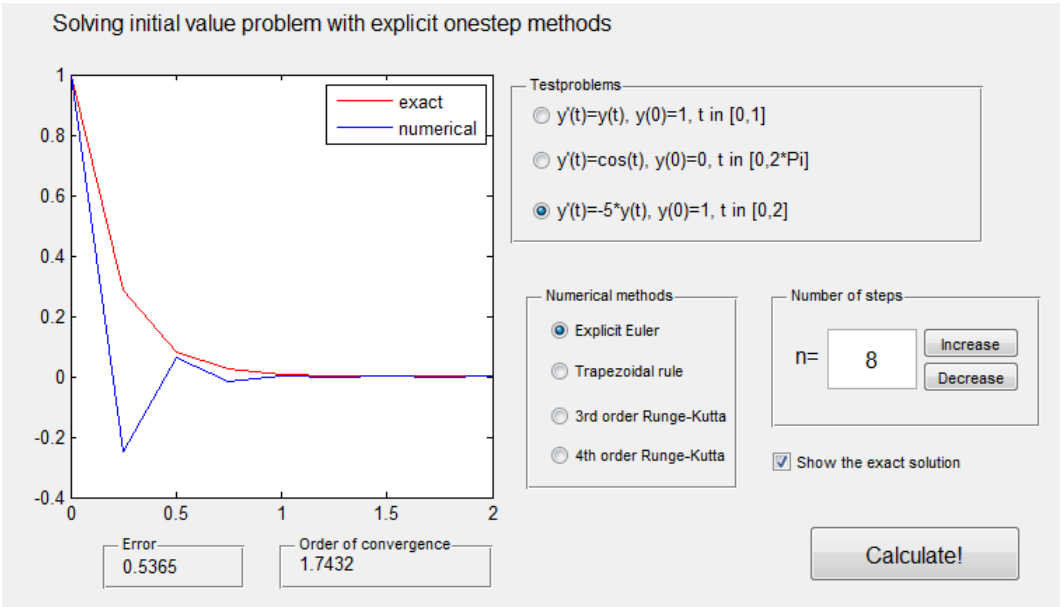

Figure 2.1: The image on the screen of the interactive program for several explicit one-step methods

the trapezoidal method, the 3rd order Runge–Kutta method and the 4th order Runge–Kutta method. We can select the discretization step size by giving the parameter n, which is the number of the partitions of the interval. Pushing "Calculate" we get the result. The result is given graphically, and the error (in maximum norm) is also indicated. By increasing n the order of the convergence is also shown.

Some further details of explicit Runge–Kutta methods can be found under the link

`http://math.fullerton.edu/mathews/n2003/RungeKuttaMod.html`

Here one can also see the behavior of the above investigated numerical methods as applied to the differential equation $u' = 1 - t\sqrt[3]{u}$.

2.4.3 Implicit Runge–Kutta method

As we have seen, the explicit Runge–Kutta methods are defined by the special (strictly lower triangular) matrix $\mathbf{B} \in \mathbb{R}^{m\times m}$ and the vector $\boldsymbol{\sigma} \in \mathbb{R}^m$. A further and quite natural generalization is obtained by neglecting the requirement for the matrix $\mathbf{B}$. Namely, we introduce the following notion.

Definition 2.4.8. *Let $\boldsymbol{B} \in \mathbb{R}^{m\times m}$ be an arbitrary matrix, $\boldsymbol{\sigma} \in \mathbb{R}^m$ some given vector, and $\boldsymbol{a} = \boldsymbol{Be}$. The numerical method defined by these parameters, i.e.,*

the method

$$\begin{aligned} k_1 &= f\left(t_i + a_1 h, y_i + h[b_{11}k_1 + b_{12}k_2 + \cdots + b_{1m}k_m]\right), \\ k_2 &= f(t_i + a_2 h, y_i + h[b_{21}k_1 + b_{22}k_2 + \cdots + b_{2m}k_m]), \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ k_m &= f(t_i + a_m h, y_i + h[b_{m1}k_1 + b_{m2}k_2 + \cdots + b_{mm}k_m]), \end{aligned} \tag{2.142}$$

$$y_{i+1} = y_i + h(\sigma_1 k_1 + \sigma_2 k_2 + \ldots + \sigma_m k_m), \tag{2.143}$$

is called Runge–Kutta method. When $\boldsymbol{B}$ *is not strictly lower triangular, i.e., there exists an element* b_{ij} *such that* $i \geq j$ *and* $b_{ij} \neq 0$*, then the method* (2.142)-(2.143) *is called implicit Runge–Kutta method. When* $\boldsymbol{B}$ *is not a strictly lower triangular matrix, but it is lower triangular, i.e.,* $b_{ij} = 0$ *for any* $i > j$*, then the method* (2.142)- (2.143) *is called diagonally implicit Runge–Kutta method.*

Remark 2.4.7. For the diagonally implicit Runge–Kutta method the algorithm is the following:

$$\begin{aligned} k_1 &= f(t_i + a_1 h, y_i + h b_{11} k_1), \\ k_2 &= f(t_i + a_2 h, y_i + h[b_{21}k_1 + b_{22}k_2]), \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ k_m &= f(t_i + a_m h, y_i + h[b_{m1}k_1 + b_{m2}k_2 + \cdots + b_{mm}k_m]), \end{aligned} \tag{2.144}$$

$$y_{i+1} = y_i + h(\sigma_1 k_1 + \sigma_2 k_2 + \ldots + \sigma_m k_m), \tag{2.145}$$

Remark 2.4.8. The main difference between them is as follows. For the explicit Runge–Kutta method we can compute the intermediate values k_i directly, by substituting the previously defined values $k_1, k_2, \ldots, k_{i-1}$. For the diagonally implicit Runge–Kutta method we cannot compute k_i directly, because, according to (2.144), k_i is present on the left side, too. Therefore, for its computation we have to solve an algebraic equation. Typically, for the implicit Runge–Kutta method the situation is the same, however, the computation of k_i becomes more difficult, because it is contained not only in the formula for the definition of k_i but also in all equations of (2.142). This means that for the definition of the values k_i we have to solve a system of nonlinear algebraic equations for m unknown values. Hence the implicit methods (DIRK and IRK) require much more computational work. However, these methods have good stability properties, which make them very beneficial for solving some special problems (like stiff ones). The other benefit of the implicit Runge–Kutta methods is that usually they are more accurate than the explicit ones.

Remark 2.4.9. The implicit Runge–Kutta method can be written as

$$k_j = f(t_i + a_j h, y_i + h \sum_{s=1}^{m} b_{js} k_s), \quad j = 1, 2, \dots m, \tag{2.146}$$

$$y_{i+1} = y_i + h \sum_{j=1}^{m} \sigma_j k_j. \tag{2.147}$$

For the diagonally implicit Runge–Kutta method the only difference is that the formula (2.146) is changed into the formula

$$k_j = f(t_i + a_j h, y_i + h \sum_{s=1}^{j} b_{js} k_s), \quad j = 1, 2, \dots m. \tag{2.148}$$

In the sequel we introduce some important implicit Runge–Kutta methods (including also the special case of diagonally implicit Runge–Kutta methods) by giving their Butcher tableau. We will see that to write these methods in the usual form of one-step methods (i.e., to give the function Φ) is more difficult than for the explicit methods.

- Let the Butcher tableau of a method be given by

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \tag{2.149}$$

This is a diagonally implicit Runge–Kutta method, which yields the following algorithm:

$$\begin{aligned} k_1 &= f(t_i + h, y_i + h k_1) \\ y_{i+1} &= y_i + h k_1. \end{aligned} \tag{2.150}$$

The realization of this method means the following. In the first step we solve the equation for the unknown k_1[16], then we substitute this value into the second formula. Let us find the function Φ for this method. From the second formula we have $hk_1 = y_{i+1} - y_i$. Therefore, by its substitution into the first equation $k_1 = f(t_i + h, y_i + (y_{i+1} - y_i)) = f(t_i + h, y_{i+1})$. From the second equation we have $y_{i+1} = y_i + hf(t_i + h, y_{i+1})$, therefore, $\Phi(h, t_i, y_i, y_{i+1}) = f(t_i + h, y_{i+1})$. Hence, the method with (2.149) is the well-known implicit Euler method. As we already know, this is a first order method.

[16]For the solution usually we use some iterative method (as the Newton method).

- Let the numerical method be given by

$$\begin{array}{c|c} 0.5 & 0.5 \\ \hline & 1 \end{array} \tag{2.151}$$

This is also a diagonally implicit Runge–Kutta method, which results in the following computational algorithm:

$$\begin{aligned} k_1 &= f(t_i + 0.5h, y_i + 0.5hk_1) \\ y_{i+1} &= y_i + hk_1. \end{aligned} \tag{2.152}$$

Let us find the function Φ for this method, too. From the second equation we have $hk_1 = y_{i+1} - y_i$. By its substitution into the first equation, we obtain $k_1 = f(t_i + 0.5h, y_i + 0.5(y_{i+1} - y_i)) = f(t_i + 0.5h, 0.5(y_i + y_{i+1}))$. Hence $\Phi(h, t_i, y_i, y_{i+1}) = f(t_i + 0.5h, 0.5(y_i + y_{i+1}))$ and the corresponding one-step method (in the usual compact form) is

$$y_{i+1} = y_i + hf(t_i + 0.5h, 0.5(y_i + y_{i+1})). \tag{2.153}$$

The diagonally implicit Runge–Kutta method (2.153) is called *implicit midpoint rule*. The order of the method can be defined by analyzing the expression (local approximation error) of the form

$$g_i = u(t_{i+1}) - u(t_i) - hf(t_i + 0.5h, 0.5(u(t_i) + u(t_{i+1}))).$$

Let us define the order in h of g_i. By developing into Taylor series, we get

$$u(t_{i+1}) - u(t_i) = hu'(t_i) + \frac{h^2}{2}u''(t_i) + \mathcal{O}(h^3),$$

and similarly

$$\begin{aligned} &f(t_i + 0.5h, 0.5(u(t_i) + u(t_{i+1}))) \\ &= f(t_i + 0.5h, u(t_i) + 0.5hu'(t_i) + \mathcal{O}(h^2)) \\ &= f(t_i, u(t_i)) + 0.5h\partial_1 f(t_i, u(t_i)) + 0.5hu'(t_i)\partial_2 f(t_i, u(t_i)) + \mathcal{O}(h^2) \\ &= f(t_i, u(t_i)) + \frac{h}{2}[\partial_1 f(t_i, u(t_i)) + f(t_i, u(t_i))\partial_2 f(t_i, u(t_i))] + \mathcal{O}(h^2). \end{aligned}$$

When we substitute these value into the formula for g_i, then taking into account the second relation in (2.4) we get $g_i = \mathcal{O}(h^3)$. This means that the implicit midpoint rule is a second order method.

- We consider the method with the Butcher tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 0.5 & 0.5 \\ \hline & 0.5 & 0.5 \end{array} \tag{2.154}$$

This is a two-stage implicit Runge–Kutta method with the following computational algorithm:

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + h, y_i + 0.5k_1 + 0.5k_2) \\ y_{i+1} &= y_i + 0.5hk_1 + 0.5hk_2. \end{aligned} \tag{2.155}$$

From the third expression we get $0.5h(k_1 + k_2) = y_{i+1} - y_i$. Substituting this expression and the first relation into the second formula, we get $k_2 = f(t_i + h, y_i + (y_{i+1} - y_i)) = f(t_i + h, y_{i+1})$. Now, putting this value of k_2, and k_1 from the first equation into the third equation, we obtain $\Phi(h, t_i, y_i, y_{i+1}) = 0.5[f(t_i, y_i) + f(t_i + h, y_{i+1})]$. Hence, the numerical method with the Butcher tableau (2.154) is the well-known second order trapezoidal method.

For the following two-stage methods we give only their Butcher tableau.

- We define the method, when in the numerical integration we choose the Gaussian basic points. This implies the following method:

$$\begin{array}{c|cc} \frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\ \frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & 0.5 & 0.5 \end{array} \tag{2.156}$$

  The method (2.156) is a two-stage, fourth order implicit Runge–Kutta method.

- We define another two-stage implicit Runge–Kutta method as follows

$$\begin{array}{c|cc} \frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\ 1 & \frac{3}{4} & \frac{1}{4} \\ \hline & \frac{3}{4} & \frac{1}{4} \end{array} \tag{2.157}$$

  The method (2.157) is called *two-stage Radau method*. We can show that the order of accuracy of this implicit Runge–Kutta method is 3.

During the consideration of the accuracy of the above implicit Runge–Kutta methods we have seen that the accuracy of the methods (p) can be greater then the number of the stages (m). For instance, the accuracy of the one-stage trapezoidal method has second order accuracy, the two-stage (2.156) method with Gaussian basic points has fourth order, etc. This means that the case $p > m$ is also possible for the implicit methods. (The reason is that

the implicit Runge–Kutta methods have more free parameters.) We can show that the upper bound for the accuracy of some m-stage implicit Runge–Kutta method is $p \leq 2m$.

The methods with $p = 2m$, are called *implicit Runge–Kutta method with maximum accuracy*. Hence, the implicit Euler method, the implicit midpoint rule, and the method (2.156) with Gaussian points are all methods of maximum accuracy.

We enclose an interactive animation, called *onestep_imp.exe*. The program can be downloaded from
`http://www.cs.elte.hu/~faragois/nummod_jegyzet_prog/`.

The image of this program on the screen can be seen in Figure 2.2.

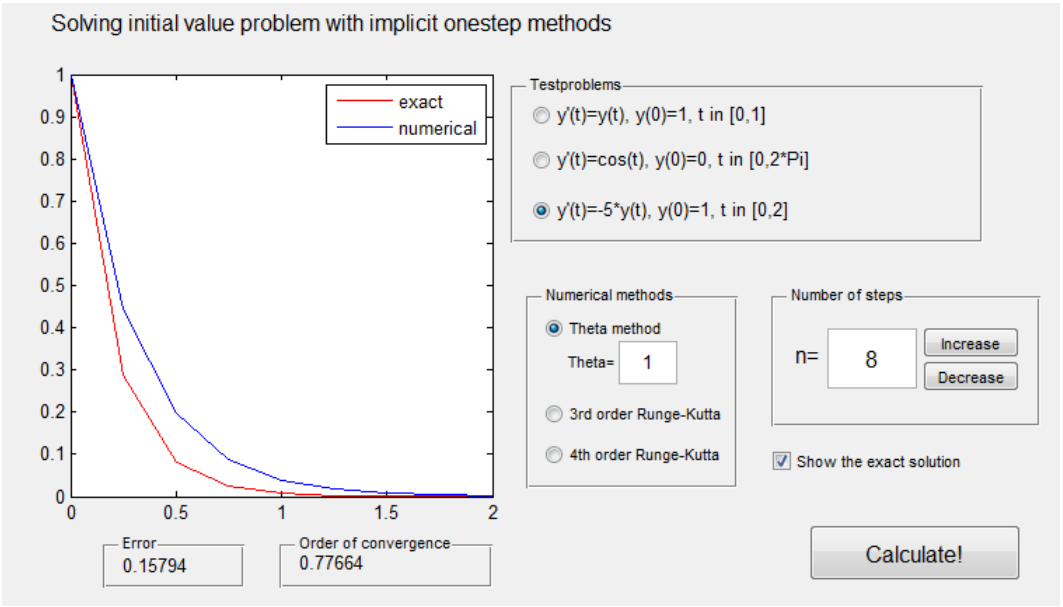

Figure 2.2: The image on the screen of the interactive program for several implicit one-step methods

Alternatively, this program suggests three initial-value problems as test problems. The chosen numerical methods can be the theta-method, where the value of theta can be chosen, the 3rd order Runge–Kutta method and the 4th order Runge–Kutta method. We can select the discretization step size by giving the parameter n, which is the number of the partitions of the interval. Pushing "Calculate" we get the result. The result is given graphically, and the error (in maximum norm) is also indicated. By increasing n the order of the convergence is also shown.

2.4.4 Embedded Runge–Kutta methods

Some differential equations may have solutions that change rapidly in some time intervals and change relatively slowly in other time intervals. (The Van der Pol equation of the form

$$u'' + u = \mu(1 - u^2)u'$$

can serve as an example.) It appears that we should not keep the step size h in the Runge–Kutta methods as a constant. Rather, we should only use a small h when the solution rapidly changes with time. A numerical method that automatically selects the step size in each step is an adaptive method. A class of adaptive methods for solving differential equations is the so-called embedded Runge–Kutta methods. An embedded Runge–Kutta method uses two ordinary Runge–Kutta methods for comparing the numerical solutions and selecting the step size. Moreover, the two methods in an embedded method typically share the evaluation of f (remember that we are solving the equation $u' = f(t, u)$). Therefore, the required computational effort is minimized.

We consider a third order explicit Runge-Kutta method, having the form

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + h, y_i + hk_1) \\ k_3 &= f(t_i + 0.5h, y_i + 0.25h(k_1 + k_2)) \\ y_{i+1} &= y_i + \frac{h}{6}(k_1 + 4k_3 + k_2). \end{aligned} \tag{2.158}$$

The costs of this method are mainly related to the calculation of the intermediate parameters k_1, k_2 and k_3. This requires three evaluations of the function f. With the above k_1 and k_2, we can use the second order Runge–Kutta method to get a less accurate solution at the point $t = t_{i+1}$:

$$\widetilde{y_{i+1}} = y_i + 0.5h(k_1 + k_2).$$

(See formula (2.109).)

Although we are not going to use $\widetilde{y_{i+1}}$ as the numerical solution at the time level t_{i+1}, we can still use it to compare with the third order solution y_{i+1}. If their difference is too large, we reject the solution and use a smaller step size h to repeat the calculation. If their difference is small enough, we will accept y_{i+1}. But we also use this information to suggest a step size for the next step. We must specify a small number ε (called error tolerance) to control the error

for selecting the step size. The difference between $\widetilde{y_{i+1}}$ and y_{i+1} is

$$\delta = |\widetilde{y_{i+1}} - y_{i+1}| = \frac{h}{3}|k_1 - 2k_3 + k_2|.^{17}$$

To understand the formula for changing the step size, we consider the first step and we use the exact solution $u(t_1)$ at this point. Using the knowledge of the order of the local approximation error, we have

$$u(t_1) - \tilde{y_1} = C_1 h^3 + \mathcal{O}(h^4),$$

$$u(t_1) - y_1 = C_2 h^4 + \mathcal{O}(h^5).$$

In these formulas the constants C_1 and C_2 are unknown, as we have already seen before. Hence,

$$y_1 - \tilde{y_1} = C_1 h^3 + \mathcal{O}(h^4),$$

i.e., we have

$$\delta = |C_1| h^3 + \mathcal{O}(h^4).$$

This implies that

$$\delta \approx |C_1| h^3. \tag{2.159}$$

Although we do not know C_1, the above relationship allows us to design a step size selection method based on the specified error tolerance ε. When $\delta \leq \varepsilon$, we accept y_1, otherwise, we reject this approximated value y_1 and repeat this step. For calculating the new values y_1 and $\tilde{y_1}$, we use the information obtained above. Namely, we should have such an h_{new}, for which the estimate

$$\delta_{new} \approx |C_1| h_{new}^3 < \varepsilon \tag{2.160}$$

holds. Comparing (2.159) and (2.160), we get

$$\frac{|C_1| h_{new}^3}{|C_1| h^3} < \frac{\varepsilon}{\delta},$$

which results in the upper bound for the new step size in the form

$$h_{new} < h \left(\frac{\epsilon}{\delta}\right)^{1/3}. \tag{2.161}$$

To satisfy the above inequality, we can use

$$h_{new} := 0.9h \left(\frac{\epsilon}{\delta}\right)^{1/3} \tag{2.162}$$

[17] When the method is applied to the Cauchy problem of a system, then the absolute values are replaced by some vector norm.

to reset the step size. Now, if for this new step size we get $\delta \leq \varepsilon$, we accept y_1 for the approximation at the point $t_{1,new} = t_0 + h_{new}$, otherwise we repeat the process. This gives rise to the possibility to increase the step size when the original h is too small (so that δ is much smaller than ϵ).

The above embedded Runge–Kutta method can be summarized as follows.

Algorithm 2.4.1. *We solve the equation $u' = f(t,u)$ from t_0 to $t^\star$ with error tolerance ε and initial condition $u(t_0) = u_0$.*

1. *We initialize $t = t_0$, $y = u_0$, ε and the initial step size h.*

2. *while* $< t \leq t^\star$

$$k_1 = f(t,y)$$
$$k_2 = f(t+h, y+hk_1)$$
$$k_3 = f(t+0.5h, y+0.25h(k_1+k_2))$$
$$\delta = \frac{h}{3}|k_1 - 2k_3 + k_2|$$
if $\delta \leq \varepsilon$ *then*
$$y = y + \frac{h}{6}(k_1 + 4k_3 + k_2)$$
$$t = t + h$$
output t, y.
end if
$$h = 0.9h\left(\frac{\epsilon}{\delta}\right)^{1/3}.$$
end

Note that the formula for resetting h is outside the "if...end if" loop. That is, whether the calculation is accepted or not, h will always be changed.

Remark 2.4.10. The Butcher tableau of the embedded Runge–Kutta methods has the following property: for the lower and the higher order methods the matrix $\mathbf{B}$ is the same, and the algorithm of the two methods are different only in the vector $\boldsymbol{\sigma}$. We also note that not every Runge–Kutta method has its embedded pair.

2.4.5 One-step methods on the test problem

We have considered several numerical methods, which differ in their accuracy (order) and in the way of their computational realization (explicit / implicit). In this section we classify them by another point of view, giving new characterization of the methods.

Let $\lambda < 0$ be some given real number and consider the *test equation* of the form

$$u' = \lambda u, \quad u(0) = 1. \tag{2.163}$$

(Clearly, the exact solution of (2.163) is the function $u(t) = \exp(\lambda t)$.) We solve this problem with explicit and implicit Euler methods. The errors are listed in Table 2.6.

	$\lambda = -9$		$\lambda = -99$		$\lambda = -999$	
h	EE	IE	EE	IE	EE	IE
0.1	3.07e-01	1.20e-01	3.12e+09	9.17e-02	8.95e+19	9.93e-03
0.01	1.72e-02	1.60e-02	3.62e-01	1.31e-01	2.38e+95	9.09e-02
0.001	1.71e-03	1.60e-03	1.90e-02	1.75e-02	3.67e-01	1.32e-01
0.0001	1.66e-04	1.65e-04	1.78e-03	1.68e-03	1.92e-02	1.76e-02
0.00001	1.66e-05	1.66e-05	1.82e-04	1.18e-04	1.83e-03	1.83e-03

Table 2.6: The error in maximum norm by different values of λ for the test problem (2.163), solved by the explicit Euler method and implicit Euler method.

We can notice that for $\lambda = -9$ the behavior of both methods corresponds to our theoretical results. (This means that by decreasing h the error is of first order.) However for the cases $\lambda = -99$ and $\lambda = -999$ this is not the case: for the starting values of h (i.e., bigger h) the explicit Euler method does not approximate the exact solution, however, the implicit Euler method does it suitably. By the further decrease of h, both methods well approximate the exact solution, and the order also corresponds to the theory (first order). Hence we can conclude that for these problems the implicit Euler method is suitable for any choice of h, however the explicit Euler method is "good" only for $h < h_0$ with some suitably chosen $h_0 > 0$. But this latter requirement results in some serious trouble for several problems (like the numerical solution of partial differential equations): when the values of λ are further decreased, then the bound h_0 is so small for the explicit Euler method, under which

1. h_0 is close the "computer epsilon" (computer zero), hence the computer realization of the method is impossible;

2. when h_0 is greater than the "computer epsilon", but small, and we want to get the numerical solution on the interval $[0, t^\star]$ with $t^\star$ of moderate size, then we need to execute too many steps ($n_{t^\star} \approx t^\star/h_0$). Therefore, in some cases, the required computational time grows dramatically. On the other hand, due to the huge number of arithmetic operations, the errors

arising at each step can accumulate, and this effect spoils the quality of the approximation. (In several situations the error even tends to infinity.)

The reason of this phenomenon is as follows. Let us examine the numerical solution at some fixed time level. For the numerical solution of the test problem (2.163) with the explicit Euler method we have the recursion

$$y_{i+1} = (1 + h\lambda)y_i, \quad i = 0, 1, \ldots, \quad y_0 = 1,$$

while with the implicit Euler method we get the recursion of the form

$$y_{i+1} = \frac{1}{1 - h\lambda} y_i, \quad i = 0, 1, \ldots, \quad y_0 = 1, \ldots$$

We can write these formulas in a unified way: they have the form of the following one-step recursion

$$y_{i+1} = R(h\lambda)y_i, \tag{2.164}$$

where the function $R(h\lambda)$ is defined by the applied numerical method. On the other hand, for the exact solution of the test problem (2.163) (which is the function $u(t) = \exp(\lambda t)$) we have

$$u(t_{i+1}) = \exp(h\lambda)u(t_i). \tag{2.165}$$

Hence, the applied numerical method can be characterized by the approximation property of $R(z)$ to the function $\exp(z)$.[18]

Remark 2.4.11. Let us notice that $\exp(z) - R(z)$ is the truncation error of the numerical method on the test problem, because using the relations (2.164) and (2.165), for the truncation error (defined in (2.82)) we have

$$g_i(h) = u(t_{i+1}) - R(h\lambda)u(t_i) = (\exp(h\lambda) - R(h\lambda))u(t_i).$$

Since the exact solution $u(t)$ is bounded, therefore the order of the consistency can be defined by the order of the expression $(\exp(h\lambda) - R(h\lambda))$. This means that in case of a p-th order consistent method we have $\exp(h\lambda) - R(h\lambda) = \mathcal{O}(h^{p+1})$. Since for the explicit Euler method $R_{EE}(z) = 1 + z$, and for the implicit Euler method $R_{IE}(z) = 1/(1 - z)$, while for the trapezoidal method $R_{tr}(z) = (1 + z/2)/(1 - z/2)$, therefore one can easily check the validity of the relations $\exp(z) - R_{EE}(z) = \mathcal{O}(h^2)$, $\exp(z) - R_{IE}(z) = \mathcal{O}(h^2)$, $\exp(z) - R_{tr}(z) = \mathcal{O}(h^3)$, which correspond to our previous results.

[18]The function $R(z)$ is called *stability function* of the numerical method.

The solution of the test problem for $\lambda < 0$ is bounded for $t > 0$. Hence, only such a numerical method can approximate the solution adequately which also possesses this property. This means the following requirement for the method (2.164):

$$|R(h\lambda)| \leq 1. \tag{2.166}$$

Since

$$|R_{EE}(h\lambda)| \leq 1 \quad \Leftrightarrow h \leq 2/(-\lambda), \tag{2.167}$$

therefore, for the explicit Euler method the exact condition for h_0 the condition $h \leq h_0$ is $h_0 = 2/(-\lambda)$. However, for implicit Euler method we have

$$|R_{IE}(h\lambda)| \leq 1 \ \text{ for any } h > 0. \tag{2.168}$$

The relations (2.167) and (2.168) explain why the explicit Euler method and the implicit Euler method have different behavior in Table 2.6 for different values of h.

We emphasize that the convergence of a numerical method is related to the behavior of the method as $h \to 0$. Hence, in case of convergence we only know that for *sufficiently small values of* h the numerical solution is close to the exact solution. However, it does not say anything about the solution on some *fixed mesh*. Therefore, numerical methods with stability function $R(z)$ for which the numerical solution well approximates the exact solution are of special interest. This is the motivation of the following notion.

Definition 2.4.9. *The set of complex numbers $z \in \mathbb{C}$ for which the condition*

$$|R(z)| \leq 1 \tag{2.169}$$

is true is called stability domain of the numerical method. *We say that some numerical method is* A-stable *when its stability domain contains the left complex half-plain $\mathbb{C}^- = \{z \in \mathbb{C} : Re z < 0\} \subset \mathbb{C}$.*

Therefore, A-stability means that for any complex number $z = a + ib$ with $a < 0$ the relation $|R(z)| \leq 1$ holds. One can easily show that the implicit Euler method is absolutely stable. (However, this property is not true for the explicit Euler method, since the condition (2.169) does not even hold on the real axis $\mathbb{R}^- \subset \mathbb{C}^-$.)

The stability domains of the explicit Euler method, the implicit Euler method, and the trapezoidal method are shown in Figures 2.3, 2.4, and 2.5.

For the explicit Runge–Kutta methods the stability domains are illustrated in Figures 2.6 and 2.7. The Figure 2.6 shows the stability domain of the first and second order explicit Runge–Kutta method, and Figure 2.7 illustrates the stability domains of the RK methods of the order $1, 2, 3$ and 4.

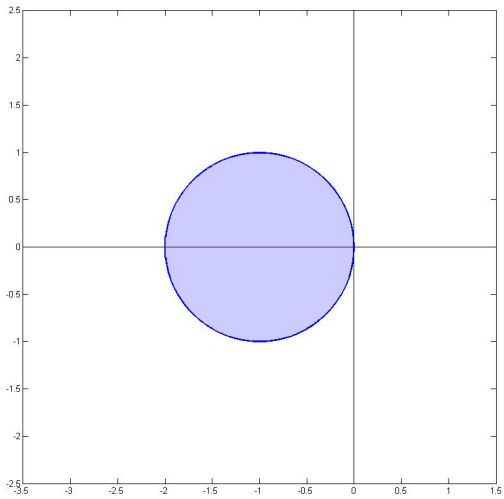

Figure 2.3: The stability domain of the explicit Euler method.

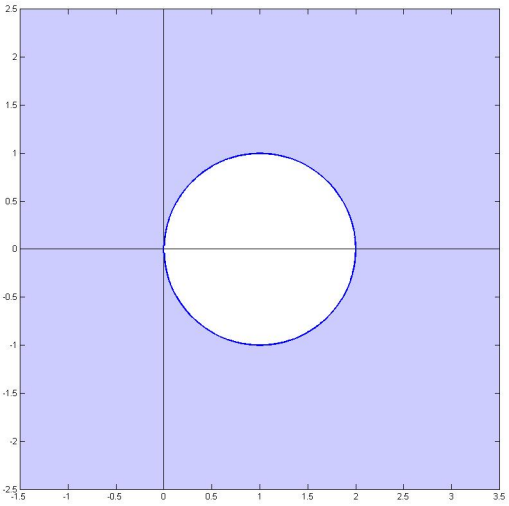

Figure 2.4: The stability domain of the implicit Euler method.

Remark 2.4.12. Let us consider the trapezoidal method, which can be writ-

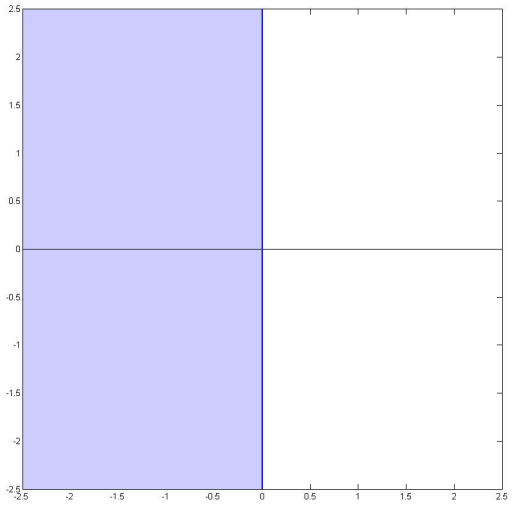

Figure 2.5: The stability domain of the trapezoidal method.

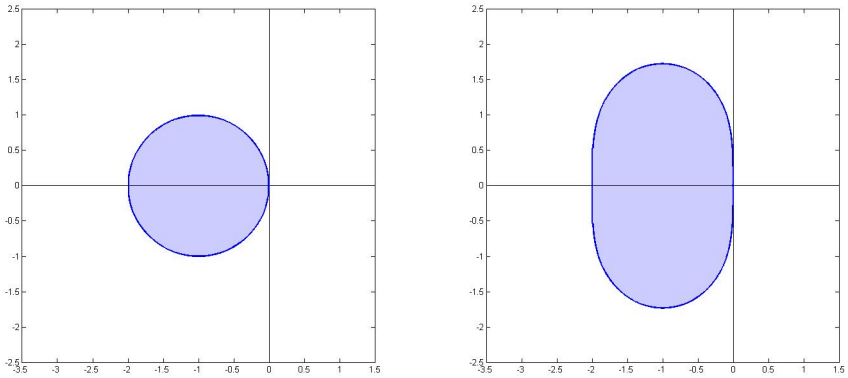

Figure 2.6: The stability domain of the first and second order explicit Runge–Kutta methods.

ten in the form

$$y_{i+1} = R_{tr}(h\lambda)y_i = \frac{1 + \lambda h/2}{1 - \lambda h/2} y_i.$$

As we have shown, this is a second order method. We can also show easily that for arbitrary $h > 0$ and $Re\lambda < 0$ the inequality $|R_{tr}(h\lambda)| \leq 1$ holds. (Hence, it is absolutely stable, too.) However, on the test problem we can

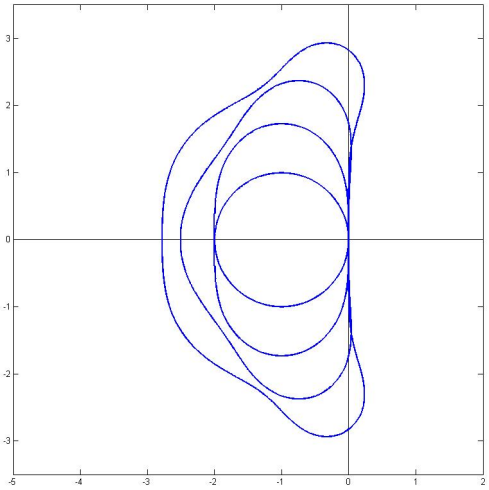

Figure 2.7: The stability domain of explicit Runge–Kutta methods of order $1-4$.

observe that the method is not good enough. Namely, when $h > 2/(-\lambda)$, then $R_{tr}(h\lambda) \in (-1, 0)$. This yields that on a mesh of such a mesh-size the sign of the numerical solution is changing at each step. Therefore the values of the numerical solution y_i are decreasing in absolute value, but they are oscillating, too. This property contradicts to the fact that the exact solution is strictly monotonically decreasing.

Chapter 3

Multistep numerical methods

In the previous sections we have considered one-step methods, i.e., such numerical methods where the value of the approximation to the exact solution is defined by the approximation of the solution at the previous mesh-point. In the sequel we generalize this approach in such a way that the new value of the approximation is defined by not only one but several previous approximations. Such methods are called multistep methods.

Further let m ($m \geq 1$) denote the number of the mesh-points at which the approximations are taken into account for the definition at the new mesh-point. Such a multistep method is called m-step method. (The one-step methods can be considered as a special case of multistep methods with the choice $m = 1$.)

Next we show on two simple examples how can these methods derived by developing the solution of the Cauchy problem (2.1)-(2.2) into Taylor series around suitably chosen points.

Example 3.0.10. *Clearly,*

$$\begin{aligned} u(t_{i-1}) =& u(t_i) - hu'(t_i) + \frac{h^2}{2}u''(t_i) + \mathcal{O}(h^3), \\ u(t_{i-2}) =& u(t_i) - 2hu'(t_i) + \frac{4h^2}{2}u''(t_i) + \mathcal{O}(h^3). \end{aligned}$$

Therefore.

$$3u(t_i) - 4u(t_{i-1}) + u(t_{i-2}) = 2hu'(t_i) + \mathcal{O}(h^3) = 2hf(t_i, u(t_i)) + \mathcal{O}(h^3).$$

Hence, by using the notation $f_i = f(t_i, y_i)$ we can define the numerical method as follows

$$y_i - \frac{4}{3}y_{i-1} + \frac{1}{3}y_{i-2} = \frac{2}{3}hf_i, \quad i = 2, 3, \ldots. \tag{3.1}$$

As we can see, the method (3.1) *is two-step, implicit, and it has second order consistency.*

Example 3.0.11. *We develop the exact solution and its derivative into Taylor series at the point t_{i-1}. Hence we have the relations*

$$\begin{aligned} u(t_i) =& u(t_{i-1}) + hu'(t_{i-1}) + \frac{h^2}{2}u''(t_{i-1}) + \mathcal{O}(h^3), \\ u'(t_{i-2}) =& u'(t_{i-1}) - hu''(t_{i-1}) + \mathcal{O}(h^2). \end{aligned}$$

From the second relation we have $hu''(t_{i-1}) = u'(t_{i-1}) - u'(t_{i-2}) + \mathcal{O}(h^2)$. Substituting this expression into the first formula, we obtain

$$u(t_i) = u(t_{i-1}) + \frac{h}{2}[3u'(t_{i-1}) - u'(t_{i-2})] + \mathcal{O}(h^3).$$

Based on this relation, we define the numerical method

$$y_i - y_{i-1} = h[\frac{3}{2}f_{i-1} - \frac{1}{2}f_{i-2}], \quad i = 2, 3, \ldots, \tag{3.2}$$

which is two-step, explicit and has second order consistency.

3.1 The consistency of general linear multistep methods

Following the above examples, we can define the linear multistep methods in a general form, too.

Definition 3.1.1. *Let $a_0, a_1, \ldots a_m$ and $b_0, b_1, \ldots b_m$ be given numbers. The iteration of the form*

$$a_0y_i + a_1y_{i-1} + \cdots + a_my_{i-m} = h[b_0f_i + b_1f_{i-1} + \ldots + b_mf_{i-m}], \quad i = m, m+1, \ldots, \tag{3.3}$$

is called linear, m-step method.

In the sequel we always assume that $a_0 \neq 0$, otherwise we are not able to define the unknown approximation y_i from the known values $y_{i-m}, y_{i-m+1}, \ldots, y_{i-1}$. According the notation $f_i = f(t_i, y_i)$, the method (3.3)) is *explicit* when $b_0 = 0$, and it is *implicit* when $b_0 \neq 0$. We fix some linear multistep method by specifying the parameters a_k and b_k $(k = 0, 1, \ldots, m)$. For instance, the numerical method (3.1) is obtained by choosing the values $m = 2$, $a_0 = 1$, $a_1 = -4/3$, $a_2 = 1/3$, and $b_0 = 2/3$, $b_1 = 0$, $b_2 = 0$. When there are given two m-steps linear multistep methods with the parameters (a_k, b_k) and $(a_k^\star, b_k^\star)$, and there exists a parameter $\beta \neq 0$ such that $a_k = \beta a_k^\star$ and $b_k = \beta b_k^\star$ for any $k = 0, 1, \ldots, m$, then from the given initial values both methods generate the

same result. These methods will be considered as the same method. Hence, the $2m+2$ parameters $((a_k, b_k), \quad k = 0, 1, \ldots m)$ which define some linear multistep method, should be a'priori fixed. This is the reason why we always assume that

$$a_0 = 1. \tag{3.4}$$

Further we define the consistency and the order of consistency of the linear multistep method (3.3). The local approximation error has the form

$$g_i(h) = \sum_{k=0}^{m} [a_k u(t_{i-k}) - h b_k f(t_{i-k}, u(t_{i-k}))]. \tag{3.5}$$

Based on the equation (2.1) we have $u'(t_{i-k}) = f(t_{i-k}, u(t_{i-k}))$. Therefore

$$g_i(h) = \sum_{k=0}^{m} \left[a_k u(t_{i-k}) - h b_k u'(t_{i-k})\right]. \tag{3.6}$$

We develop the functions on the right side of (3.6) into Taylor series at the point $t = t_i$ up to order p and $p-1$, respectively. Then

$$\begin{aligned} u(t_{i-k}) =& u(t_i - kh) = u(t_i) - khu'(t_i) + \frac{1}{2!}k^2h^2u''(t_i) \\ &+ \ldots + (-1)^p \frac{1}{p!}k^p h^p u^{(p)}(t_i) + \mathcal{O}(h^{p+1}), \\ u'(t_{i-k}) =& u'(t_i - kh) = u'(t_i) - khu''(t_i) \\ &+ \ldots + (-1)^{p-1} \frac{1}{(p-1)!}k^{p-1}h^{p-1}u^{(p)}(t_i) + \mathcal{O}(h^p). \end{aligned}$$

Substituting these expressions into (3.6), for the local approximation error we obtain

$$g_i(h) = d_0 u(t_i) + h d_1 u'(t_i) + h^2 d_2 u''(t_i) + \ldots + h^p d_p u^p(t_i) + \mathcal{O}(h^{p+1}), \tag{3.7}$$

where

$$\begin{aligned} d_0 &= \sum_{k=0}^{m} a_k, \\ d_1 &= -\sum_{k=0}^{m} (k a_k + b_k) \\ d_2 &= \sum_{k=0}^{m} (\frac{1}{2}k^2 a_k + k b_k), \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ d_p &= (-1)^p \sum_{k=0}^{m} (\frac{1}{p!}k^p a_k + \frac{1}{(p-1)!}k^{p-1} b_k). \end{aligned} \tag{3.8}$$

Some linear multistep methods are of consistency order p, when $g_i(h) = \mathcal{O}(h^{p+1})$, which results in the condition $d_0 = d_1 = \ldots = d_p = 0$. Using (3.8), we obtained the following statement.

Theorem 3.1.2. *The linear multistep method of the form* (3.3) *has p-th order consistency when for the parameters of the method the following conditions are satisfied:*

$$\begin{aligned} a_0 = 1, \quad \sum_{k=0}^{m} a_k = 0 \\ \frac{1}{j}\sum_{k=0}^{m} k^j a_k + \sum_{k=0}^{m} k^{j-1} b_k = 0, \quad j = 1, 2, \ldots, p. \end{aligned} \tag{3.9}$$

Using the above statement, the condition of the consistency of a linear multistep method can also be formulated.

Corollary 3.1.3. The linear multistep method of the form (3.3) is consistent if and only if for the parameters of the method the conditions

$$\begin{aligned} a_0 = 1, \quad \sum_{k=0}^{m} a_k = 0 \\ \sum_{k=0}^{m} k a_k + \sum_{k=0}^{m} b_k = 0 \end{aligned} \tag{3.10}$$

are satisfied.

Remark 3.1.1. Obviously, the condition of consistency (3.10) means the following: a linear multistep method is consistent if for its parameters the relations

$$\begin{aligned} 1 + a_1 + \ldots + a_m = 0 \\ (a_1 + 2a_2 + \ldots + m a_m) + (b_0 + b_1 + \ldots + b_m) = 0. \end{aligned} \tag{3.11}$$

hold. Moreover, the method is consistent in the order $p \geq 2$ when in addition to the condition (3.11) the conditions

$$\frac{1}{j}\sum_{k=1}^{m} k^j a_k + \sum_{k=1}^{m} k^{j-1} b_k = 0, \quad j = 2, 3, \ldots, p \tag{3.12}$$

are also true. Particularly, this means that an m-steps linear multistep method is consistent in second order when besides the condition of consistency (3.11), the condition

$$\frac{1}{2}\sum_{k=1}^{m} k^2 a_k + \sum_{k=1}^{m} k b_k = 0 \tag{3.13}$$

is also satisfied. One can easily verify that for the numerical method (3.1) (where $m = 2$, $a_0 = 1$, $a_1 = -4/3$, $a_2 = 1/3$, and $b_0 = 2/3$, $b_1 = 0$, $b_2 = 0$) these conditions are satisfied.

What is the maximum accuracy (order of consistency) of a linear multistep method, given in the form (3.3)? As we know, in the general form of the method there are $2m + 1$ parameters ($a_1, a_2, \ldots, a_m$ and $b_0, b_1, \ldots, b_m$), which can be chosen arbitrarily. However, to achieve p-th order consistency, these parameters have to satisfy $p + 1$ conditions. Hence we get $p \leq 2m$. When the method is explicit, i.e., $b_0 = 0$, then we have by one fewer free parameters in the method, so, the maximum order is also less by one.

Summarizing, we have the following statement.

Theorem 3.1.4. *The maximum consistency order of the m-stage implicit linear multistep method is $2m$, while for the explicit linear multistep method it is $2m - 1$.*

Remark 3.1.2. Instead of the condition (3.4) (i.e., $a_0 = 1$), which guarantees the uniqueness if the linear multistep method, other conditions can also be given. One of such conditions is

$$\sum_{k=0}^{m} b_k = 1. \tag{3.14}$$

This condition guarantees that for the method (3.3), rewritten in the form

$$\frac{a_0 y_i + a_1 y_{i-1} + \cdots + a_m y_{i-m}}{h} = b_0 f_i + b_1 f_{i-1} + \ldots + b_m f_{i-m},$$

the right side exactly approximates the $f =$ constant function. From (3.10) we can observe that under the assumption (3.14) the condition of consistency is

$$a_0 = 1, \quad \sum_{k=0}^{m} a_k = 0 \tag{3.15}$$

$$\sum_{k=1}^{m} k a_k = -1. \tag{3.16}$$

From the condition (3.12) we can see that, aiming at getting order $p \geq 2$, the conditions are the following:

$$\sum_{k=1}^{m} k^{j-1} \left(k a_k + j b_k\right) = 0, \quad j = 2, 3, \ldots, p. \tag{3.17}$$

We can see that in such a formulation of the linear multistep method we have $2m+2$ unknowns, and $p+2$ conditions. Hence (quite naturally) we get again $p \leq 2m$, and the maximum order $p = 2m$ can be defined in the following way.

1. For the unknown parameters $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_m$ we solve the system of algebraic equations (3.16)–(3.17), consisting of $2m$ equations.

2. Then, from the conditions (3.14) and (3.15), by the formula

$$a_0 = -\sum_{k=1}^{m} a_k = 0, \quad b_0 = 1 - \sum_{k=1}^{m} b_k \tag{3.18}$$

   we define the parameters a_0 and b_0, respectively.

For the analysis of linear multistep methods it is useful to introduce the following polynomials:

$$\varrho(\xi) = \sum_{k=0}^{m} a_k \xi^{m-k} \tag{3.19}$$

$$\sigma(\xi) = \sum_{k=0}^{m} b_k \xi^{m-k}. \tag{3.20}$$

Definition 3.1.5. *The polynomials of the m-th degree $\varrho(\xi)$ and $\sigma(\xi)$, given in* (3.19) *and* (3.20)*, are called first and second characteristic polynomials of the linear multistep method.*

From the condition (3.10), after some calculation, we get the following statement [1, 12].

Theorem 3.1.6. *A linear multistep method is consistent when for its characteristic polynomials the relations*

$$\varrho(1) = 0, \quad \varrho'(1) = \sigma(1) \tag{3.21}$$

are true.

3.2 The choice of the initial conditions and the stability

As we have seen, some m-step linear multistep methods assume the knowledge of the approximations at the first m mesh-points of the mesh, i.e., the values $y_0, y_1, \ldots, y_{m-1}$ are considered as given.

However, from the initial condition (2.2) we know only the value y_0. The other initial conditions to the linear multistep method $y_1, \ldots, y_{m-1}$ are usually defined by the use of some suitably chosen one-step method. (This means that the accuracy of the one-step method coincides with the accuracy of the applied linear multistep method. Typically, this one-step method is some Runge–Kutta method with sufficiently high accuracy.) We emphasize that the orders of the linear multistep method and the applied one-step method should be equal, because if the one-step method has lower order, we lose the order of accuracy of the linear multistep method.)[1]

We consider the convergence of linear multistep methods. As we have already seen in the case of one-step methods, consistency in itself is not enough for the convergence. In the sequel (without proof) we give the conditions under which the convergence holds.

For a consistent method the value $\xi_k = 1$ is a root of the first characteristic equation. (See condition (3.21).) The following definition tells us what other roots are allowed.

Definition 3.2.1. *We say that a linear multistep method* satisfies the root criterion *when for the roots $\xi_k \in \mathbb{C}$ $(k = 1, 2, \ldots, m)$ of the characteristic equation $\varrho(\xi) = 0$ the inequality $|\xi_k| \leq 1$ holds, and the roots with the property $|\xi_k| = 1$ are single.*

The next theorem shows that the root criterion for a linear multistep method means its stability.

Theorem 3.2.2. *Assume that a linear multistep method is consistent, and the root criterion is valid. Then the method is convergent, i.e., for any fixed point $t^\star \in (0, T)$ we have the relation $y_n \to u(t^\star)$ as $h \to 0$, where $nh = t^\star$.*

Remark 3.2.1. The following example illustrates the role of the root criterion: when it is destroyed, then the method is not stable (and hence, not convergent). We consider the two-step, explicit linear multistep method of the form

$$y_i + 4y_{i-1} - 5y_{i-2} = h(4f_{i-1} + 2f_{i-2}).$$

We can easily check that the method has maximum accuracy, i.e, its order is $p = 2m - 1 = 3$. The first characteristic polynomial is $\varrho(\xi) = \xi^2 + 4\xi - 5 = (\xi - 1)(\xi + 5)$. The roots are obviously $\xi_1 = 1$ and $\xi_1 = 5$, which means that the root criterion is not satisfied. Let us consider the equation $u' = 0$

[1] We note that in the program package the approximations y_k $(k = 1, 2, \ldots m - 1)$ are defined by a suitably chosen $k - 1$-step method.

with the initial condition $u(0) = 0$. (Clearly, the exact solution is the function $u(t) = 0$.) We solve numerically this problem using the above method. We choose $y_0 = 0$ and $y_1 = \varepsilon$. (When we compute the first approximation y_1 by some one-step method, then the result will be a good approximation to the exact solution, which means that it will be close to zero. Hence, ε is a small number, close to zero.) Then the use of the numerical method results in the following approximations:

$$\begin{aligned} y_2 &= -4y_1 = -4\varepsilon \\ y_3 &= -4y_2 + 5y_1 = 21\varepsilon \\ y_4 &= -4y_3 + 5y_2 = -104\varepsilon \quad \text{etc.} \end{aligned}$$

We can observe that the numerical results are increasing, and the values of the approximations are not bounded. Hence, the considered numerical method is not convergent.[2]

We note that the above example shows that the root criterion is a necessary condition of the convergence. However, the question of its sufficiency is yet to be answered. (I.e., does the root criterion guarantee the convergence?) The answer to this question is negative, i.e., the root criterion in itself is not enough for the convergence. This means that even under the root criterion some numerical methods result in bad approximations due to the errors arising in the computations. In these methods the problem is that their characteristic equation has more than one different roots with absolute values equal to one.

To this aim, we introduce the following notion.

Definition 3.2.3. *We say that a linear multistep method is* strongly stable*, when it satisfies the root criterion, and $\xi_k = 1$ is the only root with the property $|\xi_k| = 1$.*

As an example, we consider the Milne method, having the form

$$y_i - y_{i-2} = \frac{h}{3}(f_i + 4f_{i-1} + f_{i-2}).$$

The roots of its characteristic polynomial are $\xi_{1,2} = \pm 1$. Hence, the root criterion is true, but the method is not strongly stable. Therefore, the usage of this method is not recommended. For a strongly stable linear multistep method we recall the famous result, given by G. Dahlquist, which shows that the maximum order of such methods are rather restrictive.

[2]We emphasize that in this example ε can be considered as some small perturbation of the exact value, i.e., even putting for y_1 its exact value $u(h)$, (which is theoretically equal to zero), due to the computer representation, we have only some approximate value, which can be considered as $y_1 = \varepsilon$.

Theorem 3.2.4. *The maximum order of an m-step linear multistep method is* $p = m + 1$.

For one-step methods we have already shown that the convergence does not give any information about its adequate behavior on some fixed mesh. (The convergence ensures that for sufficiently small mesh-size the numerical result is close to the exact solution. However, as we have seen, this small step size can be unrealistic.) To avoid this situation, we have defined the absolute stable methods. (See Definition 2.4.9.)

For a linear multistep methods the following question is quite natural: Which linear multistep methods are absolutely stable? The answer to this question shows that for these methods it is difficult to guarantee the absolute stability. Namely, according to the first and second order barrier, given by Dahlquist, we have

- Explicit linear multistep methods are not A-stable.
- The maximum order of an A-stable linear multistep method is two.

These barriers yield a serious problem in the application of linear multistep methods.

3.3 Some linear multistep methods and their analysis

In the sequel we consider and analyze some linear multistep methods, which are typically used in the applications.

3.3.1 Adams methods

One of the most important linear multistep methods are obtained when in the general formula (3.3) the parameters a_i are defined as follows:

$$a_0 = 1, \quad a_1 = -1, \quad a_2 = a_3 = \ldots = a_m = 0. \tag{3.22}$$

Such methods are called *Adams method*s. (For instance, the method given in (3.2) is an Adams method.) In the Adams methods the parameters $b_0, b_1, \ldots, b_m$ can be chosen arbitrarily, i.e., they are free parameters.

There is a difference between the Adams methods with $b_0 = 0$ and $b_0 \neq 0$: in the first case the method is explicit, while in the second case it is implicit.

Definition 3.3.1. *The Adams methodwith $b_0 = 0$ is called* Adams–Bashforth method*, and the Adams method with $b_0 \neq 0$ is called* Adams–Moulton method .

The condition of consistency and the order of consistency of an Adams method can be defined directly from the conditions (3.11) and (3.12).

Theorem 3.3.2. *An Adams method is consistent if and only if the condition*

$$b_0 + b_1 + \ldots + b_m = 0 \tag{3.23}$$

is satisfied. Moreover, the method is consistent of order $p \geq 2$, when besides the condition (3.23) *the conditions*

$$\sum_{k=1}^{m} k^{j-1} b_k = \frac{1}{j}, \quad j = 2, 3, \ldots, p \tag{3.24}$$

are also satisfied.

The maximum order of an m-step Adams method can also be easily defined: for the Adams–Moulton method it is $p = m+1$, while for the Adams–Bashforth method $p = m$.

We note that the first characteristic polynomial of the Adams method is

$$\varrho(\xi) = \xi - 1. \tag{3.25}$$

Since it has the only root $\xi = 1$, therefore for this method the root criterion is always satisfied, moreover, such methods are strongly stable, too. This proves the following statement.

Theorem 3.3.3. *The Adams method is convergent with the order, equal to the order of the consistency.*

In Table 3.1 we give the values of the coefficients b_k for the maximum order Adams–Bashforth methods, up to order six. (For the easier interpretation, in the table we write the coefficient b_k not in fraction form.)

In this table, according to the theoretical results, $p = m$. The case $m = 1$ results in the explicit Euler method. For the two-step method ($m = 2$) we obtain the already known method (3.2). The Adams–Bashforth methods are not A-stable, since they are explicit (see the first Dahlquist barrier). Moreover, their stability domain is relatively small. For the case $m = 1$ (i.e., for the explicit Euler method) the stability domain is the set of complex numbers with the property $|1 + z| \leq 1$, i.e., the circle with the center $(-1, 0)$ and unit radius

p	m	b_k	1	2	3	4	5	6
1	1	b_k	1					
2	2	$2b_k$	3	-1				
3	3	$12b_k$	23	-16	5			
4	4	$24b_k$	55	-59	37	-9		
5	5	$720b_k$	1901	-2774	2616	-1274	251	
6	6	$1440b_k$	4277	-7923	9982	-7298	2877	-475

Table 3.1: The coefficients in the Adams-Bashfort multistep methods

on $\mathbb{C}$. (C.f. (2.167).) With an increase of m this domain gets narrower. Hence, these methods are not suitable for solving problems where absolute stability is required. This motivates the usage of the Adams–Moulton methods.

For the Adams–Bashforth method we enclose an interactive animation, called *multistep_exp.exe*. The program can be downloaded from `http://www.cs.elte.hu/~faragois/nummod_jegyzet_prog/`.

The image of this program on the screen can be seen in Figure 3.1.

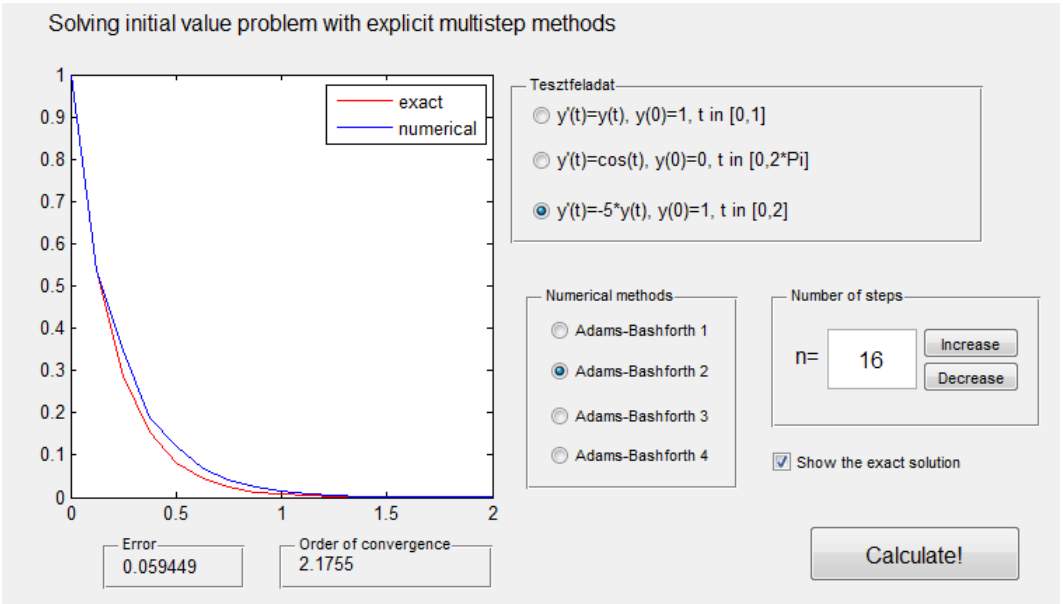

Figure 3.1: The image on the screen of the interactive program for several explicit linear multistep methods

p	m	b_k	0	1	2	3	4	5
1	1	b_k	1					
2	1	$2b_k$	1	1				
3	2	$12b_k$	5	8	-1			
4	4	$24b_k$	9	19	-5	1		
5	5	$720b_k$	251	646	-264	106	-19	
6	6	$1440b_k$	475	1427	-798	482	-173	27

Table 3.2: The coefficients in the Adams-Moulton multistep methods

Alternatively, this program suggests three initial-value problems as the test problems. The chosen numerical methods can be the explicit Adams-Bashfort method of first, second, third and fourth order. We can select the discretization step size by giving the parameter n, which is the number of partitions of the interval. Pushing "Calculate" we get the result. The result is given graphically, and the error (in maximum norm) is also indicated. By increasing n the order of the convergence is also shown.

In Table 3.2, up to the order six, we give the values of the coefficients b_k for the maximum order Adams–Moulton methods. (For the easier interpretation, as before, in the table we write the coefficient b_k not in fraction form.)

The first method ($m = 1$, $b_1 = 0$) results in the well-known implicit Euler method. Since this method of first order, it is not a maximum order method. The order of the other methods are maximum, which means that, according to the theory, it is $p = m + 1$. The second method ($m = 1$, $\beta_1 \neq 0$) is already known: this means the trapezoidal method, given by the formula (2.68). We note that the stability domain of an Adams–Moulton methods are larger than the stability domain of Adams–Bashforth methodof the same order. In the above Adams–Moulton methods only the first and the second methods (i.e., the implicit Euler method and a trapezoidal method) are A-stable, the others are not. (This is the consequence of the second Dahlquist barrier.)

For the Adams–Moulton method we enclose an interactive animation, called *multistep_imp.exe*. The program can be downloaded from
`http://www.cs.elte.hu/~faragois/nummod_jegyzet_prog/`.

The image of this program on the screen can be seen in Figure 3.2.

Alternatively, this program suggests three initial-value problems as test problems. The chosen numerical methods can be the explicit Adams–Moulton method of second and third order, or the Curtis–Hirschfeld method of second

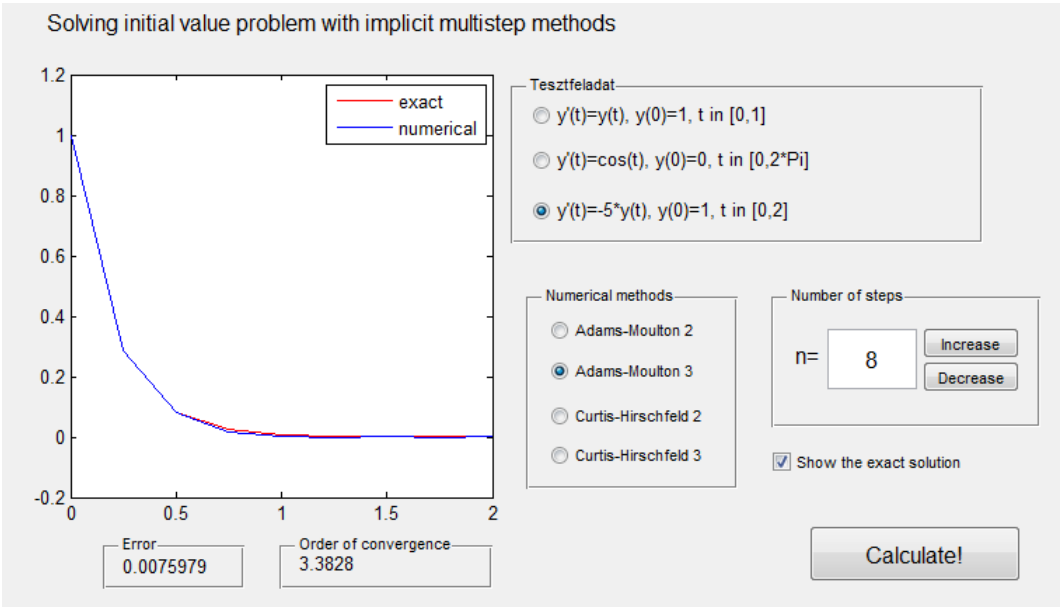

Figure 3.2: The image on the screen of the interactive program for several implicit linear multistep methods

and third order.[3] We can select the discretization step size by giving the parameter n, which is the number of partitions of the interval. Pushing "Calculate" we get the result. The result is given graphically, and the error (in maximum norm) is also indicated. By increasing n the order of the convergence is also shown.

Remark 3.3.1. Very often the Adams–Bashforth methods and the Adams–Moulton methods are combined in the following way. First, by using some suitably chosen Adams–Bashforth method we define the values $y_i^\star$, and these values are considered as "predicted values" at the time level t_i. Then, using some Adams–Moulton method, we improve these numerical results in the following way: on the right side of the formula, in the term $b_0 f_i$ instead of f_i we substitute the values $f_i^\star = f(t_i, y_i^\star)$, and these values are considered as "corrected values". This procedure is called "predictor-corrector" (PC) method.[4] An important property of the "predictor-corrector method is that they are explicit.
Some further theoretical results of the PC method can be found under the link `http://math.fullerton.edu/mathews/n2003/AdamsBashforthMod.html`

[3] For the Curtis–Hirschfeld methods we refer to the next subsection.
[4] Sometimes it is also called *Adams-Bashforth-Moulton method*

and for several PC methods one can see animations for the solution of the differential equation $u' = 1 - t\sqrt[3]{u}$, too.

3.3.2 Backward difference method

Besides the Adams methods, an important role is played by implicit linear multistep methods, given by the formula (3.3) with the parameter choice

$$b_0 \neq 0, \quad b_1 = b_2 = \ldots = b_m = 0. \tag{3.26}$$

Such methods are called *backward difference methods.*[5] The consistency and the order of the consistency are defined with a suitable choice of the free parameters $a_0, a_1, \ldots a_m$.

An important feature of the backward difference method is that it requires the evaluation of the function $f(t, u)$ only at one point (t_i, y_i). The backward difference methods are especially suitable for the numerical integration of initial value problems which require some special stability property from the numerical method. (See stiff problems in the next section.) Based on Remark 3.1.1, the following statement is true.

Theorem 3.3.4. *The m-step numerical integration formula of the form*

$$y_i + a_1 y_{i-1} + \cdots + a_m y_{i-m} = h b_0 f_i, \quad i = m, m+1, \ldots, \tag{3.27}$$

called backward difference method, is consistent if and only if the conditions

$$\begin{aligned} a_1 + \ldots + a_m &= -1 \\ a_1 + 2a_2 + \ldots + m a_m &= -b_0 \end{aligned} \tag{3.28}$$

are satisfied. Moreover, the order of consistency of the method is $p \geq 2$ when, besides the condition (3.28)*, the conditions*

$$\sum_{k=1}^{m} k^j a_k = 0, \quad j = 2, 3, \ldots, p \tag{3.29}$$

are also satisfied.

We can see that there are $p + 1$ conditions for the $m + 1$ unknowns b_0, a_1, a_2 , ..., a_m. This means that the maximum order of a backward difference method is $p = m$. In Table 3.3 we summarize the coefficients of the first six backward difference methods with maximum order.

[5]Such methods are also called *retrograde difference methods.*

p	m	b_0	a_0	a_1	a_2	a_3	a_4	a_5	a_6
1	1	1	1	-1					
2	2	$\frac{2}{3}$	1	$-\frac{4}{3}$	$\frac{1}{3}$				
3	3	$\frac{6}{11}$	1	$-\frac{18}{11}$	$\frac{9}{11}$	$-\frac{2}{11}$			
4	4	$\frac{12}{25}$	1	$-\frac{48}{25}$	$\frac{36}{25}$	$-\frac{16}{25}$	$\frac{3}{25}$		
5	5	$\frac{60}{137}$	1	$-\frac{300}{137}$	$\frac{300}{137}$	$-\frac{200}{137}$	$\frac{75}{137}$	$-\frac{12}{137}$	
6	6	$\frac{60}{147}$	1	$-\frac{360}{147}$	$\frac{450}{147}$	$-\frac{400}{147}$	$\frac{225}{147}$	$-\frac{72}{147}$	$\frac{10}{147}$

Table 3.3: The coefficients in the backward difference methods

The first method ($m = 1$) is the implicit Euler method, while the methods with $m = 2, 3, 4$ are called *Curtis-Hirschfeld method of second, third and fourth order*. The first two methods ($m = 1, 2$) are A-stable, while the others are not. (This correspond to the second Dahlquist barrier.) However, their stability domain is large. [6] We note that for $m > 6$ the a backward difference methods lose their basic stability property, necessary for convergence (so called zero-stability), therefore backward difference methods with higher than sixth order are not applicable.

3.4 Stiff problems and their numerical solution

In the previous sections we analyzed ordinary differential equations in scalar form. Now we consider the problem of the form

$$\begin{aligned} \frac{d\mathbf{u}}{dt} &= \mathbf{A}\mathbf{u}(t), \quad t \in (0, T], \\ \mathbf{u}(0) &= \mathbf{u}_0, \end{aligned} \tag{3.30}$$

where $\mathbf{A} \in \mathbb{R}^{m\times m}$ is a given matrix, $\mathbf{u}_0 \in \mathbb{R}^m$ a given vector, and $\mathbf{u} : [0, T] \to \mathbb{R}^m$ is the unknown function. The problem (3.30) is called *Cauchy problem for a linear system of ordinary differential equations*. In the next section we consider the numerical treatment of such problems.

[6] The backward difference methods for $m = 3, 4, 5, 6$ are not A-stable, which means that their stability domain does not contain the whole $\mathbb{C}^-$ complex half-plain. However, the stability domain contains the sector with the edge at the origin and angle $\pm\alpha$. Such methods are called $A(\alpha)$-stable. (Hence A-stability is $A(\alpha)$-stability with angle $\alpha = 90°$.) However, by increasing m these sectors get narrower: for $m = 3$ we have $\alpha \simeq 86°$, (i.e., it is almost A-stable), but for $m = 6$ we have $\alpha \simeq 17,8°$, only.

3.4.1 Numerical investigation of the linear systems

For simplicity of the investigation, we assume that $\mathbf{A}$ has m different eigenvalues, and hence it is diagonalizable, i.e., there exists a regular matrix $\S$ such that the matrix $\S^{-1}\mathbf{A}\S = \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues of $\mathbf{A}$ in the main diagonal.[7]

Let us introduce the new unknown function $\mathbf{w}(t) = \S^{-1}\mathbf{u}(t)$. Then the initial value problem (3.30) can be rewritten as

$$\begin{aligned} \frac{d\mathbf{w}}{dt} &= \Lambda\mathbf{w}(t), \quad t \in (0,T], \\ \mathbf{w}(0) &= \S^{-1}\mathbf{u}_0. \end{aligned} \tag{3.31}$$

We note that (3.31) is a system with m unknown functions such that it consists of m independent *scalar problems* of the form

$$w_k' = \lambda_k w_k, \quad w_k(0) = \text{given}, \ k = 1, 2, \ldots, m \tag{3.32}$$

which were already considered (c.f. (2.163)).

Consequently, we can lead the problem of the numerical solution of linear systems to the numerical solution of linear scalar problems. This means that the numerical methods are applied to the test problems of the form (3.32), where λ_k are the eigenvalues of the matrix $\mathbf{A}$. This implies that for the application of some numerical method to the linear system with matrix $\mathbf{A}$ we should examine the behavior of the method on the spectrum of the matrix on the test equations (3.32).

For the explicit Euler method we have seen that a suitable choice of the step size of the mesh is made under the condition (2.167). Therefore, in the

[7] A matrix with such a property is also called *matrix of a simple structure*. For such matrices the columns of the matrix $\S$ consist of the eigenvectors of the matrix A. This is a sufficient condition, only. The exact (necessary and sufficient) condition can be given as follows. A matrix is diagonalizable, i.e., it is similar to some diagonal matrix, when it has m linearly independent eigenvectors. Hence, if the matrix has m distinct eigenvalues, then it must have m linearly independent eigenvectors, therefore it is diagonalizable. However, it does not say that if we have less than m distinct eigenvalues, then the matrix is not diagonalizable. To verify this property is a bit complicated task, and it is connected to the geometric multiplicity of the eigenvectors. If the sum of the geometric multiplicities of all eigenvalues is exactly m, then the matrix has a set of m linearly independent eigenvectors, and hence it is diagonalizable. If for some eigenvalue the algebraic multiplicity is not equal to the geometric multiplicity, then the matrix is not diagonalizable. A matrix that is not diagonalizable is said to be defective. For defective matrices, the notion of eigenvector can be generalized to generalized eigenvectors, and that of the diagonal matrix to a Jordan form matrix.

sequence of problems (3.32) we get the bound

$$|R_{EE}(h|\lambda_k|)| \leq 1, \quad k = 1, 2, \ldots, m, \tag{3.33}$$

which yields the bound

$$h \leq 2/|\lambda_k|, \ k = 1, 2, \ldots, m. \tag{3.34}$$

Hence (3.34) results in the condition

$$h \leq \frac{2}{\max_k |\lambda_k|}. \tag{3.35}$$

We note that in this work we do not deal with differential equations of higher order. The reason is that such a system can be rewritten as a system of first order equations, hence they can be investigated within the above theory. As an example on how such a problem can be rewritten is given in the following.

We consider the initial value problem for a linear, homogeneous m-th order ordinary differential equation of the form

$$y^{(m)} + a_1 y^{(m-1)} + a_2 y^{(m-2)} + \ldots + a_{m-1} y' + a_m y = 0 \tag{3.36}$$

with the initial conditions $y(0) = c_1, \ y'(0) = c_2, \ \ldots, y^{(m-1)}(0) = c_m$.

We introduce new unknown functions $u_1, u_2, \ldots, u_m$ as follows:

$$\begin{aligned} u_1(t) &= y(t) \\ u_2(t) &= y'(t) = u_1'(t) \\ u_3(t) &= y''(t) = u_2'(t) \\ &\ldots\ldots\ldots\ldots\ldots \\ u_m(t) &= y^{m-1}(t) = u_{m-1}'(t). \end{aligned} \tag{3.37}$$

Let us differentiate both sides of the last equation. Then we get the equation

$$u_m'(t) = y^m(t) = -a_1 u_m - a_2 u_{m-1} - \ldots - a_m u_1. \tag{3.38}$$

Hence, using the definition (3.37) of the new functions, we obtain a system of linear first order ordinary differential equations for the m unknown functions, having the form

$$\mathbf{u}' = \mathbf{A}\mathbf{u},$$

where $\mathbf{u}(t) : [0, T] \to \mathbb{R}^m$ is the unknown vector-valued function with coordinates $u_i(t)$, and $\mathbf{A} \in \mathbb{R}^{m \times m}$ is the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0 & 1 \\ -a_m & -a_{m-1} & -a_{m-2} & \ldots & -a_2 & -a_1 \end{bmatrix}. \tag{3.39}$$

h	explicit Euler method	modified Euler method	RK4
0.1	6.3410^{13}	3.9910^{24}	2.8110^{41}
0.01	4.0710^{17}	1.2210^{21}	1.5310^{-19}
0.001	1.1510^{-125}	6.1710^{-108}	2.6910^{-109}

Table 3.4: Comparison of the numerical results obtained with different methods for the problem (3.40).

The initial condition is $\mathbf{u}(0) = \mathbf{c}$, where $\mathbf{c} \in \mathbb{R}^m$ is the vector with coordinates $c_1, c_2, \ldots c_m$.

In the next part we consider r a special type of problems, which causes some extra difficulties in choosing the suitable numerical method.

3.4.2 Numerical investigation of stiff problems

For the motivation, first we consider an instructive problem.

Example 3.4.1. *Consider the problem*

$$u' = -250u, \quad u(0) = 1, \tag{3.40}$$

the solution of which is the very rapidly decreasing exponential function

$$u(t) = e^{-250t}.$$

Obviously, at the point $t = 1$ the exact solution is $u(1) = e^{-250} \approx 2.6910^{-109}$.

The following table shows the results of approximating the above exact solution value $u(1)$ by using three different numerical integration methods and for various step sizes h: the explicit Euler method, the modified Euler method and the fourth order Runge–Kutta method.

The results in Table 3.4 show the difficulties arising when the considered explicit methods are applied to problem (3.40). When the step size is $h = 0.1$, the computed solution values are perplexingly large, and appear to represent an exponentially growing solution – the complete opposite of the rapidly decaying true solution. Reducing the step size beyond a critical threshold suddenly transforms the numerical solution to an exponentially decaying function. Only the fourth order RK4 method with step size $h = .001$ – and hence a total of 1000 steps – does a reasonable job at approximating the correct value of the solution at $t^\star = 1$.

Let us examine another problem leading to a system of ordinary differential equations.

Example 3.4.2. *Consider the second order, linear ordinary differential equation*

$$y'' + (\gamma + 1)y' + \gamma y = 0, \tag{3.41}$$

with the initial conditions $y(0) = 1$ *and* $y'(0) = \gamma - 2$. *(Here* γ *is some given parameter.)*

Then the corresponding first order system has the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\gamma & -(\gamma + 1) \end{bmatrix},$$

and the initial condition is given by the vector $\mathbf{c} = [1, \gamma - 2]$. The characteristic equation for the matrix $\mathbf{A}$ is

$$\det(\mathbf{A} - \lambda \mathbf{I}) = \lambda^2 + (\gamma + 1)\lambda + \gamma = 0$$

hence, the eigenvalues of the matrix $\mathbf{A}$ are $\lambda_1 = -1$ and $\lambda_2 = -\gamma$. This means that the exact solution of the problem is

$$\begin{aligned} u_1(t) &= 2\exp(-t) - \exp(-\gamma t) \\ u_2(t) &= -2\exp(-t) + \gamma \exp(-\gamma t). \end{aligned} \tag{3.42}$$

When we apply the explicit Euler method for the numerical solution, then for the choice of the mesh-size in case of $\gamma \geq 1$ we have the condition $h < 1/\gamma$. Now let us assume that $\gamma >> 1$. Then the possible chosen mesh-size is rather small. However, we can observe from the formula of the exact solution (3.42) that the functions $\exp(-\gamma t)$ in the expression of the solution practically disappears for $t \geq t_0$ already for small t_0, it did not play any noticeable role in the solution. Therefore, the exact solution is, in fact $u_1(t) \simeq -u_2(t) \simeq \exp(-t)$ on the interval $[t_0, T]$. This means that it is not necessary to choose such a small step size h on the whole interval $[0, T]$. Systems with this property are called *stiff systems*, or *stiff problems*.. Although there is no exact definition of stiff problems, for linear problems the following definition is convenient.

Definition 3.4.3. *The linear system of ordinary differential equations* (3.30) *is called stiff when the eigenvalues* λ_k $(k = 1, 2, \ldots, m)$ *of the constant coefficient matrix* $\boldsymbol{A} \in \mathbb{R}^{m \times m}$ *of the system have the following properties.*

1. $Re\lambda_k < 0$ *for each* $k = 1, 2, \ldots, m$. *(This means that the system is Lyapunov stable.)*

2. *The number defined as*

$$S = \frac{\max_k |Re\lambda_k|}{\min_k |Re\lambda_k|} \tag{3.43}$$

(so called stiffness number*) is large, i.e.,* $S >> 1$.

Typically, for a stiff system the exact solution of the initial value problem is a sum of quickly and slowly decreasing functions, and having passed some small time interval $[0, t_0]$, the solution is defined almost only by the slowly changing components. For the numerical handling of such problems explicit methods are not usually suitable, and usually A-stable methods are required. From the already considered methods we especially recommend the backward difference methods, particularly the implicit Euler method and the first two Curtis–Hirschfeld methods.

The notion of "stiffness" expresses that the original initial value problem has big stability. (Remember that stability for a Cauchy problem means that the solution continuously depends on the input data.) In the above example (3.41), as we have seen, for sufficiently large values of γ (which can be considered as an input data) the solution is independent of its concrete value. This means some extra stability property, and the simple numerical methods are not able to follow it. This is the reason why explicit methods are not suitable for such problems.

As we saw at the beginning of this section, the phenomenon of stiffness can be considered on the scalar equation, too. In order to generalize that problem, let us consider the scalar problem

$$u' = ku; \quad t > 0; \quad u(0) = u_0,$$

where k is some given constant. The solution of this problem is $u(t) = u_0 \exp(kt)$. When $k \ll 0$, then for very small t_0 the solution $u(t)$ is monotonically decreasing on the interval $(0, t_0)$ from u_0 to the computer zero, and then for the values $t > t_0$ we have $u(t) = 0$ in the computer. Hence, on the major part of the time domain the solution is independent of the initial state, which yields some extra stability property. We note that for this problem the explicit Euler method has bad behavior. E.g., taking $k = -15$ and $h = 1/4$, the numerical solution grows into infinity. The numerical solution for this problem with $h = 1/8$ remains bounded, and it follows the graph of the exact solution, however, with some oscillation.[8] However, by applying the trapezoidal method, the numerical solutions with the above parameters are already adequate.

In summary, we can conclude that, paradoxically, the larger negative k is - and hence the faster the solution tends to a trivial zero equilibrium - the more difficult and expensive the numerical integration.

[8] As we have shown, for the explicit Euler method the condition of stability is (3.35), which in our example results in the condition $h < 2/|k| = 2/15$. This condition does not hold for the first choice, but it is satisfied for the second one. For the non-oscillation of the explicit Euler method the condition is $h < 1/|k| = 1/15$, which is not valid for $h = 1/8$, either.

For some more details we refer to the link
`http://en.wikipedia.org/wiki/Stiff_equation`
but with preparing an own computer code the Reader can also check the mentioned phenomena. (See the next section, too.)

We note that the notion of stiff problem can be extended to nonlinear problems, too. In this case we consider the linearized problem, and apply the theory. This means that for nonlinear systems the role of the matrix $\mathbf{A}$ will be played by the Jacobian matrix of the system.

The stability domains of some special linear multistep methods are shown in Figures 3.3, 3.4, and 3.5. Figure 3.3 shows the stability domain of the first four Adams–Bashforth methods, while Figure 3.4 shows the stability domains of the first six Adams–Moulton methods. Figure 3.5 shows the stability domain of the first six backward difference methods.

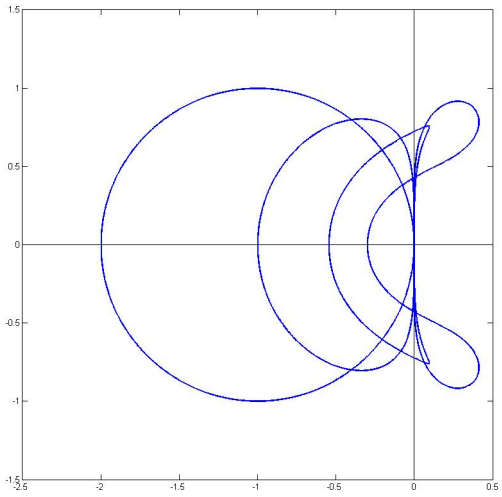

Figure 3.3: The stability domains of the first four Adams–Bashforth methods.

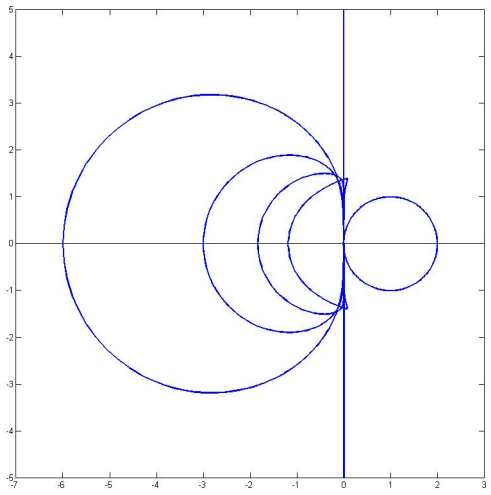

Figure 3.4: The stability domains of the first six Adams–Moulton methods.

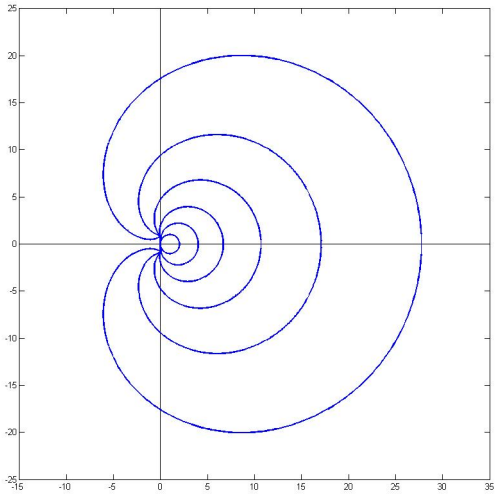

Figure 3.5: The stability domains of the first six backward difference methods.

Chapter 4

Numerical solution of initial value problems with MATLAB

The investigated numerical methods can be implemented on computer with the help of different program packages. In the following we consider MATLAB, which has several built-in routines for the different methods, however, to prepare our own code is also possible, and since this is not difficult, it is highly recommended for the Reader.[1]

4.1 Preparing MATLAB programs for some numerical methods

Let us consider the explicit Euler method and we prepare the program (a so called *m-file*) for this method. Then we can use very simply this program as a function, by giving its parameters.

First we describe those steps which are required for the implementation.

- Launch MATLAB and type the following text into the Editor:

```
function[t,y] = expeuler(diffeq, t0, y0, h, N)
t = zeros(N+1,1);
y = zeros(N+1,1);
t(1) = t0;
y(1) = y0;
for i=1:N
t(i+1) = t(i) + h;
```

[1]For several details we refer the link `http://www.masteringmatlab.com/` .

```
y(i+1) = y(i) + h * diffeq(t(i),y(i));
end
end
```

- Let us describe the program in more detail.

  In the first line of this code we gave how the program can be called. (We gave it the name expeuler.) This means that on the left side of the equality symbol = we identify the task by listing the input data and the required input parameters of the explicit Euler method. On the left side of the equality symbol = we list the output parameters in brackets $[\cdot]$. (Typically they are such values which are computed within the program and are used later for some purposes.) In our case we have five input and two output parameters. The output parameters (results) are two vectors: the first vector (t) is the vector containing the discrete time points (the mesh-points), and the second vector (y) contains the numerical solution at these points. The first input parameter $(diffeq)$ identifies the differential equation by giving the function on the right side of the differential equation (which was denoted by f). The second parameter $(t0)$ denotes the point where the initial condition is given. The third parameter $(y0)$ is the initial value at this point. The next parameter (h) is the step-size of the mesh where the numerical solution is defined. Finally, N denotes the number of the steps on this mesh. (I.e., the numerical solution is computed at the equidistant mesh-points of the interval $[t0, Nh]$.)

  In the second and third lines we give zero value to the vectors t and y, where the numerical approximations will be stored. In the next two lines we define the starting values for these vectors.

  In fact, these steps were the preparation work for the method.

  From the next line we start to give the algorithm of the method. Within a loop first we give the values of t_i, then we compute the slope of the approximation, and then we compute y_i according to the explicit Euler method.

- With this code we cannot compute directly the numerical solution of the problem, because we have to identify the function "diffeq", too. (We remind you that this function describes the right side of the equation f.)

  We will consider the example, given as

$$u'(t) = -u(t) + t + 1,$$

where $t \in [0,1]$ and $u(0) = 1$.[2] To prepare the function "diffeq", we open a new m-file, and we write the following:

```
function dydt = diffeq(t,y)
dydt = -y + t + 1;
end
```

- When both routines are ready, we can run the program. We will use the parameters $h = 0.1$, $h = 0.01$ and $h = 0.001$ on the interval $[0,1]$. In the Command window named we type the following:

  `[T1,Ye] = expeuler(@diffeq, 0, 1, 0.1, 10).`

  After pressing Enter we obtain the vectors T1 and Ye, which contain the place and the values of the approximation. If we want to draw the solution, then by giving the command

  `plot(T1,Ye)`

  we can do it, and then in a separate window we will see the plot of the numerical solution. (For $h = 0.01$ and $h = 0.001$ we should type

  `[T1,Ye] = expeuler(@diffeq, 0, 1, 0.01, 100)`

  and

  `[T1,Ye] = expeuler(@diffeq, 0, 1, 0.001, 1000)`,

  respectively.)

Remark 4.1.1. Here we have used the symbol "@". This function handle is a MATLAB value that provides a means of calling a function indirectly. We can pass function handles in calls to other functions (often called function functions). (We can also store function handles in data structures for later use.) A function handle is one of the standard MATLAB data types.

In order to get the global error of the explicit Euler method we have to compare the numerical solution with the exact solution. The solution of our problem is the function

$$u(t) = e^{-t} + t.$$

In Figures 4.1-4.3 we can see the accuracy of the method for the step sizes $h = 0.1$, $h = 0.01$ and $h = 0.001$. In accordance with the theory, by decreasing h the graph of the numerical solution approaches to the graph of the exact solution.

[2]We remind the Reader that this example was already considered in Sections 2.2 and 2.4. In this section we will solve this problem with Runge–Kutta methods, and linear multistep methods, too.

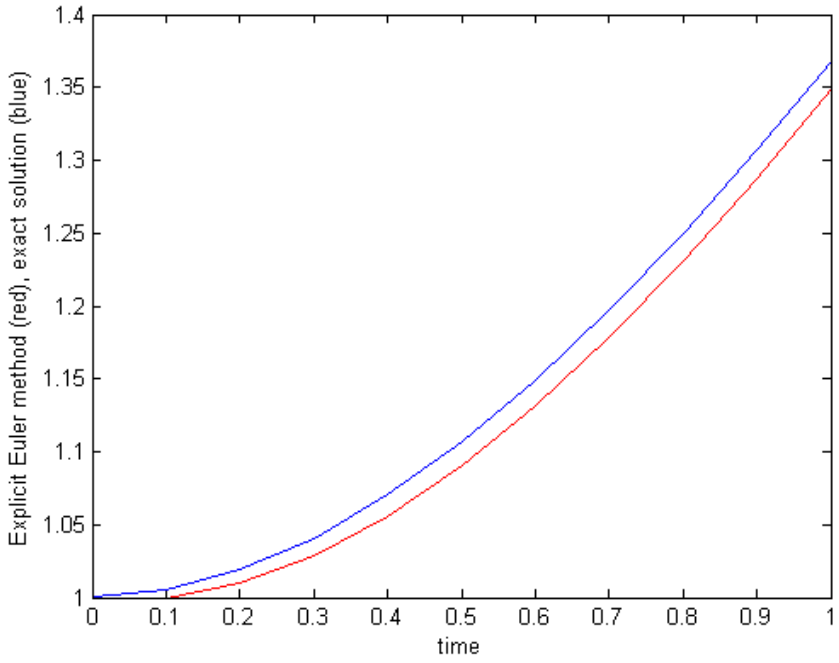

Figure 4.1: The explicit Euler method with step size $h = 0.1$.

Let us give the MATLAB program of the embedded Runge–Kutta method, given in Section 2.4.4. For the combination of the second and third order methods the MATLAB program, according to the Algorithm 2.4.1, is the following[3].

```
function[tout, yout] = embrk23(@f,t0, tfinal, u0, epsnul, h0)
%This is the embedded Runge-Kutta method using a 3rd order
% Runge-Kutta method and a 2nd order Runge-Kutta method.
% We are solving u = f(t,u)
% Input:  t0, the initial time
% tfinal, the final time
% u0, the initial condition, i.e., u(t0) = u0.
% epsnul, the small parameter for error tolerance
% h0, initial time step
% Output:  tout, a row vector for the discrete time steps
% yout, a matrix for solutions of y at various various time.
t = t0;
y = u0;
```

[3]This program requires the knowledge of f, which specifies the differential equation.

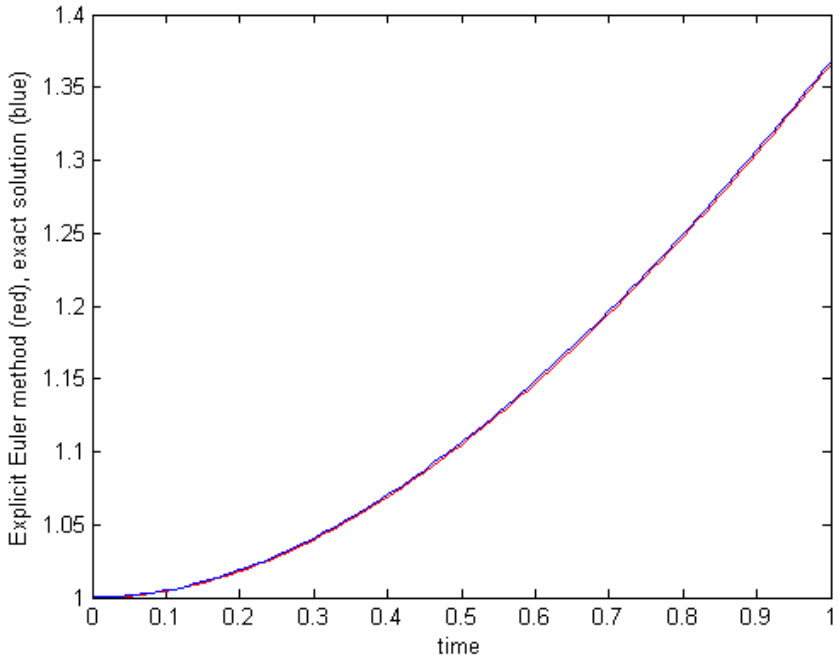

Figure 4.2: The explicit Euler method with step size $h = 0.01$.

```
h = h0;
tout = t0;
yout = y;
while t < tfinal
k1 = f(t, y);
k2 = f(t+h, y + h*k1);
k3 = f(t+h/2, y + h*(k1+k2)/4);
delta = (h/3)*norm(k1-2*k3 +k2);
if delta <= epsnul
y = y + (h/6)*(k1 + 4*k3 + k2);
t = t + h;
tout = [tout, t];
yout = [yout, y];
end
h = 0.9 * h * (epsnul/delta)^1/3;
end
```

This same problem can be solved also by several other multistep methods.

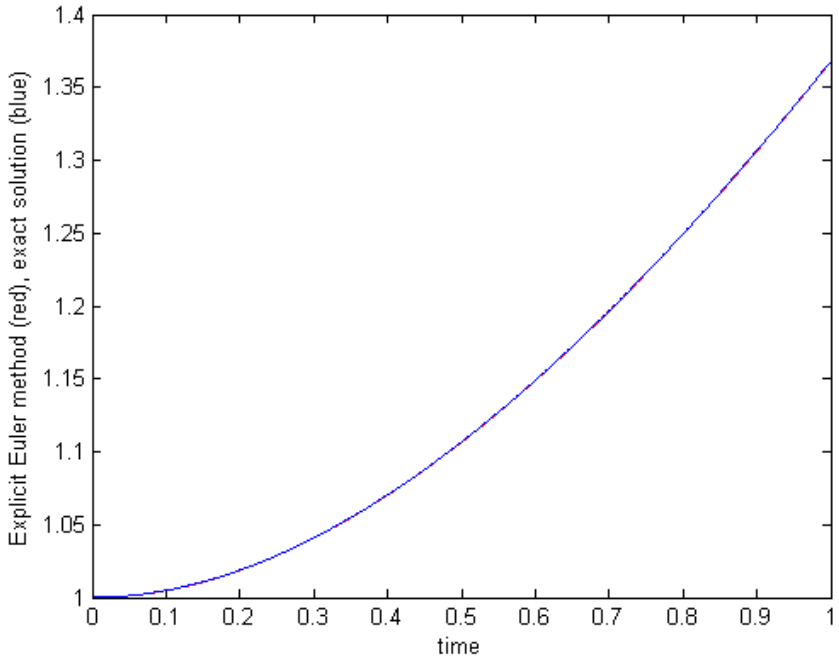

Figure 4.3: The explicit Euler method with step size $h = 0.001$.

With the choice $h = 0.2$ we solve the problem with the methods Adams-Bashforth 2, Adams-Bashforth 3, Adams-Moulton 2, Adams-Moulton 3, Backward difference method 2, and Backward difference method 3.

The results are shown on figures 4.4-4.9.

It is possible to create also animation with MATLAB. In our book we present several animations.

The animation *ee.gif* shows the behavoir of the explicit Euler method, the *ee1mee2.gif* animation uses the explicit and the modified explicit Euler methods. The animation *thetam.gif* shows the behavoir of the numerical solution on fixed mesh by varying values of θ. We have prepared also animations for the Runge-Kutta 4 method (*rk.gif*), the Adams-Bashforth 2 method (*ab2.gif*), the Adams-Bashforth 4 method (*ab4.gif*), the Adams-Moulton 2 method (*am2.gif*), the Curtis-Hirschfeld 2 method (*ch2.gif*), and comparison of explicit Euler and implicit Euler methods (*eeie.gif*).

```
../animations/ee.gif ../animations/ee1mee2.gif ../animations/thetam.
gif ../animations/rk.gif ../animations/ab2.gif ../animations/ab4.
gif ../animations/am2.gif ../animations/ch2.gif ../animations/eeie.
gif
```

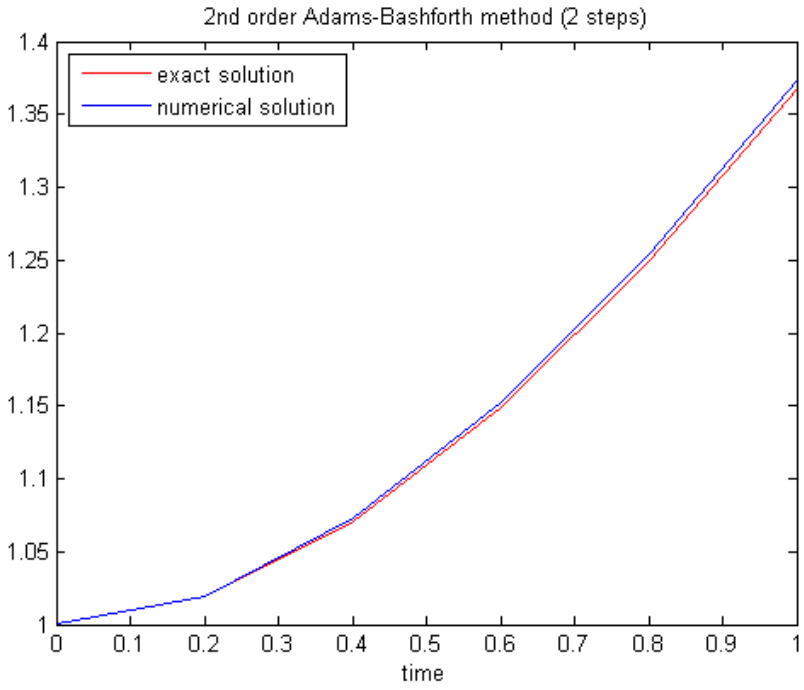

Figure 4.4: The Adams-Bashforth 2 with step size $h = 0.2$.

4.2 Built-in MATLAB programs

MATLAB is able to use built-in library, which, in fact, implement the most useful methods. These methods allow us to get the numerical solution of initial value problems efficiently with high accuracy.

One of the solvers is called ODE45, which is based on the embedded Dormand-Prince method. [4] This method is based on the combination of some fourth and fifth order Runge–Kutta methods, and the step size is chosen in such a way that the error of the combined method is the error of the fourth order method. In Table 4.1 we give the Butcher tableau of the method. (The first row for σ belongs to the fourth order method, while the second row to the fifth order method.) The routine ODE45 can be called in the same way as the routines written by us in the previous section. Namely, the form is `[T1, Y45]`

[4]Embedded methods were investigated in Section 2.4.4. We recall that these methods are based on two Runge–Kutta methods of different orders, but in their Butcher tableau $\mathbf{a}^\top$ and $\mathbf{B}$ are the same. (However, the weighting vectors $\boldsymbol{\sigma}$ are different, and therefore their orders are also different.) A suitable combination of these methods gives us the possibility to appropriately choose the varying step size in the combined method.

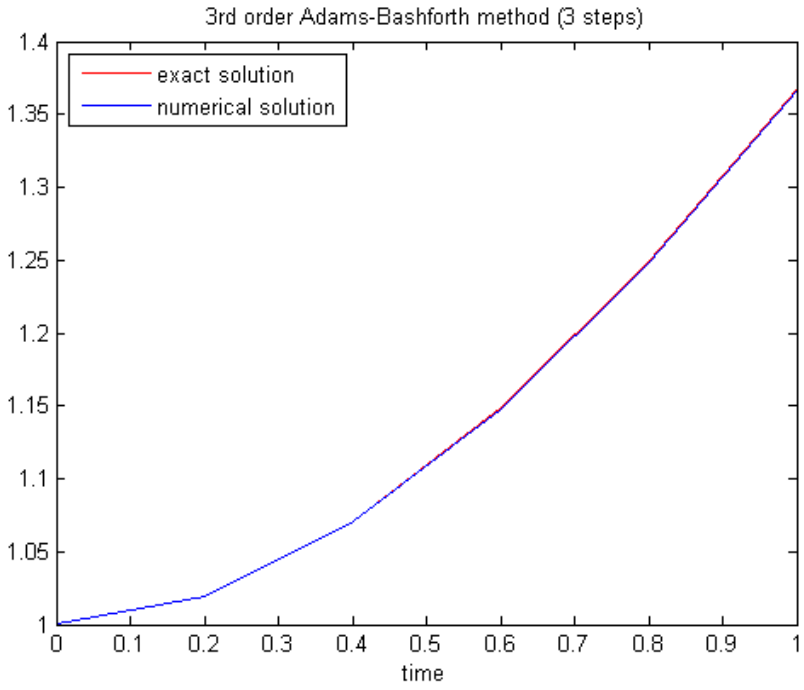

Figure 4.5: The Adams-Bashforth 3 with step size $h = 0.2$.

`= ode45(@diffeq, T1, 1)`. We have two output vectors again, which give the time points and the values of the approximations. The input parameters are the following: "diffeq" is the function which describes the right side of the equation; "T1" is the vector which gives those points where we aim the get the numerical solution, and finally we give the initial value for the problem.[5]

We checked the accuracy of ODE45 on the previous Exercise 2.13, with the step sizes $h = 0.1$ and $h = 0.01$. The corresponding results are included in Tables 4.2 and 4.3. We can see that the accuracy of the method does not decrease with decreasing the step size. The reason is that, due to the adaptive choice of the mesh-size in the method, the achievable accuracy is reached already for the first choice $h = 0.1$.

[5]We note that it is also possible to use a fourth (optional) parameter for numerical integrating. However, typically its default value is enough for our purposes. If one wants to change the default value, it is possible by using the program ODE45, and the details can be found in the help list of MATLAB.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$\frac{-56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$\frac{-25360}{2187}$	$\frac{64448}{6561}$	$\frac{-212}{729}$			
1	$\frac{9017}{3168}$	$\frac{-355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$\frac{-5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	
	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$\frac{-92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$\frac{-2187}{6784}$	$\frac{11}{84}$	0

Table 4.1: The parameters of the embedded Dormand-Prince RK in the ODE45 routine

t_i	exact solution	numerical solution	error
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	$2.9737e-010$
0.2000	1.0187	1.0187	$5.3815e-010$
0.3000	1.0408	1.0408	$7.3041e-010$
0.4000	1.0703	1.0703	$8.8120e-010$
0.5000	1.1065	1.1065	$9.9668e-010$
0.6000	1.1488	1.1488	$1.0822e-009$
0.7000	1.1966	1.1966	$1.1424e-009$
0.8000	1.2493	1.2493	$1.1814e-009$
0.9000	1.3066	1.3066	$1.2026e-009$
1.0000	1.3679	1.3679	$1.2090e-009$

Table 4.2: Results for the built-in solver ODE45 with step-size $h = 0.1$

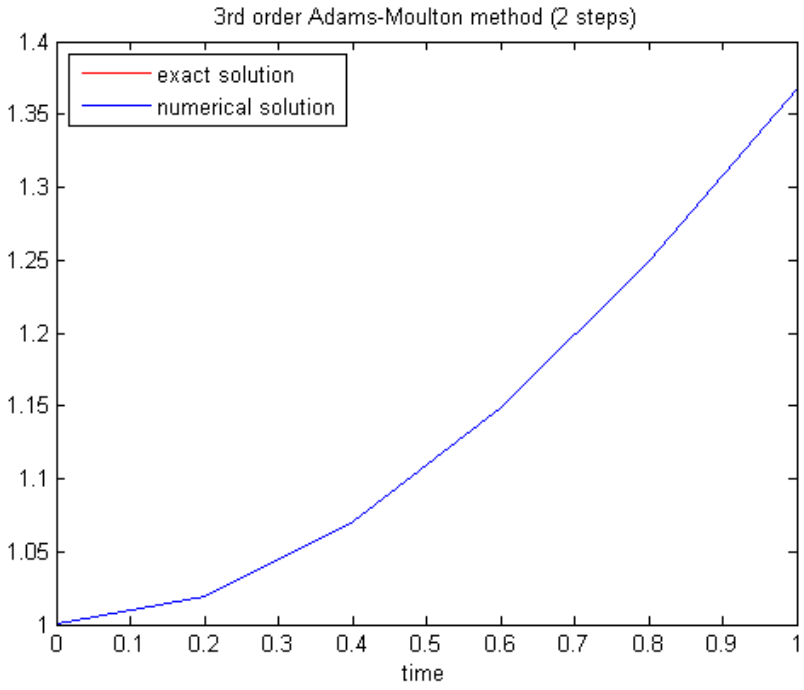

Figure 4.6: The Adams-Moulton 2 with step size $h = 0.2$.

Another built-in and widely used routine is the embedded Runge–Kutta method ODE23, which is also called Bogacki–Shampine method . This program can be called as `[T1, Y23] = ode23(@diffeq, T1, 1)`, similarly to the ODE45 method. Table 4.4 contains the Butcher tableau of the method.

We emphasize that the Bogacki–Shampine method is an explicit, $(2, 3)$-type Runge–Kutta method. This method is especially useful when we want to get the numerical solution quickly and cheaply, but with low accuracy. (Usually the routines ODE45 and ODE23 are used only for non-stiff problems.)

Testing the ODE23 routine on Exercise 2.13 for the mesh-sizes $h = 0.1$ and $h = 0.01$, we obtain Tables 4.5 and 4.6, respectively.

t_i	exact solutuion	numerical solution	error
0	1.0000	1.0000	0
0.0100	1.0000	1.0000	$2.2204e-016$
0.0200	1.0002	1.0002	$8.8940e-011$
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	$5.4080e-009$
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	$2.8278e-009$
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2493	$1.6519e-009$
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	$1.3610e-009$
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3616	$1.0920e-009$

Table 4.3: Results for the built-in solver ODE45 with step-size $h = 0.01$.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{3}{4}$	0	$\frac{3}{4}$		
1	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	
	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{4}{9}$	0
	$\frac{7}{24}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$

Table 4.4: The parameters of the embedded Bogacki-Shampine RK in the routine ODE23

t_i	exact solution	numerical solution	error
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	$4.0847e-006$
0.2000	1.0187	1.0187	$7.3920e-006$
0.3000	1.0408	1.0408	$1.0033e-005$
0.4000	1.0703	1.0703	$1.2104e-005$
0.5000	1.1065	1.1065	$1.3690e-005$
0.6000	1.1488	1.1488	$1.4865e-005$
0.7000	1.1966	1.1966	$1.5692e-005$
0.8000	1.2493	1.2493	$1.6227e-005$
0.9000	1.3066	1.3066	$1.6518e-005$
1.0000	1.3679	1.3679	$1.6607e-005$

Table 4.5: Results of the routine ODE23 for $h = 0.1$.

t_i	exact solution	numerical solution	error
0	1.0000	1.0000	0
0.0100	1.0000	1.0000	$4.1583e-010$
0.0200	1.0002	1.0002	$3.3825e-008$
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	$1.8521e-006$
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	$1.2194e-005$
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	$1.5515e-005$
⋮	⋮	⋮	⋮
0.9900	1.3616	1.3616	$1.5233e-005$
1.0000	1.3679	1.3679	$1.5087e-005$

Table 4.6: Results of the routine ODE23 for $h = 0.01$.

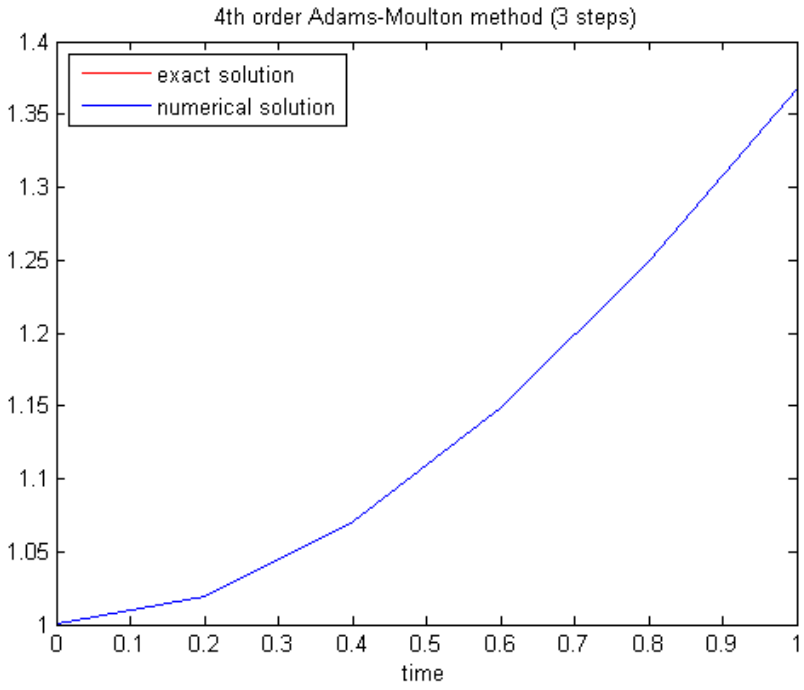

Figure 4.7: The Adams-Moulton 3 with step size $h = 0.2$.

In MATLAB we can also find built-in routines for the linear multistep methods. Such a routine is ODE113, the order of which can change from 1 to 13, and which is based on the Adams–Bashforth–Moulton method. Comparing with ODE45, we can conclude that the ODE113 routine is less accurate, but cheaper. This method is especially recommended when the evaluation of the function f is expensive.

The syntax of the routine is the usual `[T1, Y113] = ode113(@diffeq, T1, 1)` with the same output and input parameters, and the method is applicable for non-stiff problems. The accuracy of the method on the previous test problem, for the step sizes $h = 0.1$ and $h = 0.01$, can be seen in Tables 4.7 and 4.8, respectively.

t_i	exact solution	numerical solution	error
0	1.0000	1.0000	0
0.1000	1.0048	1.0048	$6.2300e-006$
0.2000	1.0187	1.0187	$1.8714e-005$
0.3000	1.0408	1.0408	$2.7885e-005$
0.4000	1.0703	1.0703	$2.1933e-005$
0.5000	1.1065	1.1065	$1.8889e-005$
0.6000	1.1488	1.1488	$1.7254e-005$
0.7000	1.1966	1.1966	$1.5668e-005$
0.8000	1.2493	1.2493	$1.4228e-005$
0.9000	1.3066	1.3066	$1.2872e-005$
1.0000	1.3679	1.3679	$1.1643e-005$

Table 4.7: Results of the routine ODE113 for $h = 0.1$.

t_i	exact solution	numerical solution	error
0	1.0000	1.0000	0
0.0100	1.0000	1.0001	$1.6625e-007$
0.0200	1.0002	1.0002	$1.4800e-007$
⋮	⋮	⋮	⋮
0.1000	1.0048	1.0048	$1.4004e-007$
⋮	⋮	⋮	⋮
0.5000	1.1065	1.1065	$1.7178e-008$
⋮	⋮	⋮	⋮
0.8000	1.2493	1.2493	$2.0090e-008$
⋮	⋮	⋮	⋮
0.9000	1.3066	1.3066	$1.8052e-008$
⋮	⋮	⋮	⋮
0.9800	1.3553	1.3553	$1.6692e-008$
0.9900	1.3616	1.3616	$1.6525e-008$
1.0000	1.3679	1.3679	$1.6360e-008$

Table 4.8: Results of the routine ODE113 for $h = 0.01$.

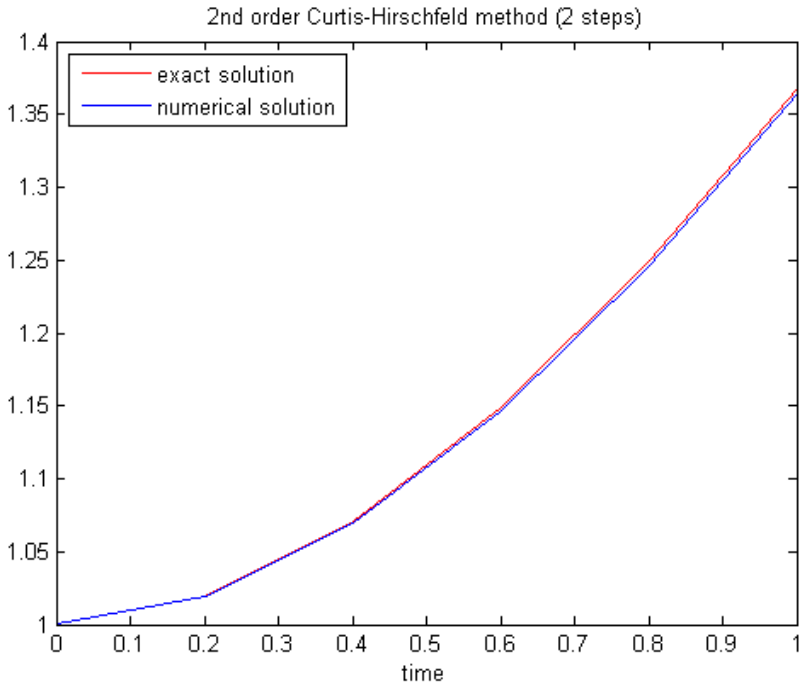

Figure 4.8: The Curtis-Hirschfeld 2 with step size $h = 0.2$.

Below we summarize the accuracy of the different methods, investigated in this part, for the same problem (2.13). We compare the errors at the time-point $t^\star = 1$ on the meshes with mesh-size $h = 0.1$ and $h = 0.01$.

method	e_{n^*}	
	$h_1 = 0.1$	$h_2 = 0.01$
explicit Euler	1.9201e-002	1.8471e-003
improved Euler	6.6154e-004	6.1775e-006
implicit Euler (more accurate)	1.7664e-002	1.8318e-003
implicit Euler	2.1537e-002	1.8687e-003
ODE45	1.2090e-009	1.0903e-009
ODE23	1.6607e-005	1.5087e-005
ODE113	1.1643e-005	1.6360e-008

Finally we note that in MATLAB there are built-in routines for stiff problems as well. An example is ODE23S, which can be called by the usual command [T1, Y23S] = ODE23S(@DIFFEQ, T1, 1)). The routines ODE123T and ODE123TB are recommended for the numerical solution of stiff problems with

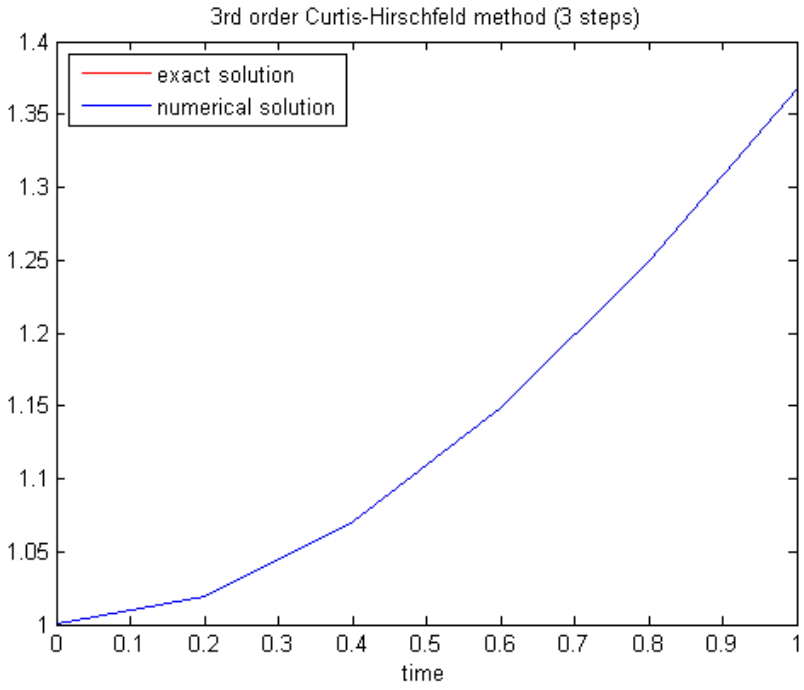

Figure 4.9: The Curtis-Hirschfeld 3 with step size $h = 0.2$.

mild stiffness numbers. (See page 81.) These methods have only moderate accuracy. The method ODE15S is based on the backward differentiation formulas, which are also called Gear methods. The algorithm uses the varying step-size formula. This method is especially applicable when ODE45 is very slow or it fails. The method ODE23S is a second order, modified one-step Rosenbrock method. Because this is a one-step method, usually it is more economic than ODE15S.
For more details we refer to [1, 13, 4]. For further MATLAB programs we refer to the internet and the help function of MATLAB.

4.3 Applications of MATLAB programs

In this section we consider some applications of MATLAB programs for certain practical problems.

4.3.1 The 360° pendulum

Normally we think of a pendulum as a weight suspended by a flexible string or cable, so that it may swing back and forth. Another type of pendulum consists of a weight attached by a light (but inflexible) rod to an axle, so that it can swing through larger angles, even making a 360° rotation if given enough velocity.

Though it is not precisely correct in practice, we often assume that the magnitude of the frictional forces that eventually slow the pendulum to a halt is proportional to the velocity of the pendulum. Assume also that the length of the pendulum is 1 m, the weight at the end of the pendulum has mass 1 kg, and the coefficient of friction is 0.5. In that case, the equations of motion for the pendulum are as follows.

$$\begin{aligned} x'(t) =& y(t), \\ y'(t) =& -0.5y(t) - 9.81 \sin x(t). \end{aligned} \tag{4.1}$$

Here t represents time in seconds, $x(t)$ represents the angle of the pendulum from the vertical in radians (so that $x = 0$ is the rest position), $y(t)$ represents the angular velocity of the pendulum in radians per second, and 9.81 is approximately the acceleration due to gravity in meters per second squared. Now here is a phase portrait of the solution with initial position $x(0) = 0$ and initial velocity $y(0) = 5$. This is a graph of x versus y as a function of t, on the time interval $0 < t < 20$. (To use MATLAB's numerical differential-equation solver ODE45, we combine x and y into a single vector x; see the online help for ODE45.)

The MATLAB code can be written as follows.

```
% MATLAB code for the 360° pendulum.
% We use the built-in program ODE45
% We solve the system x'(t) = y(t),
% y'(t) = −0.5y(t) − 9.81 sin x(t).
g = @(t, x) [x(2); -0.5*x(2) - 9.81*sin(x(1))];
[t, xa] = ode45(g, [0:0.01:20], [0 5]);
plot(xa(:, 1), xa(:, 2))
```

Recall that the x-coordinate corresponds to the angle of the pendulum and the y-coordinate corresponds to its angular velocity. Starting at $(0, 5)$, as t increases we follow the curve as it spirals clockwise toward $(0, 0)$. The angle oscillates back and forth, but with each swing it gets smaller until the pendulum is virtually at rest by the time $t = 20$. Meanwhile the angular velocity oscillates as well, taking its maximum value during each oscillation when the pendulum

is in the middle of its swing (the angle is near zero) and crossing zero when the pendulum is at the end of its swing. The corresponding picture is shown in Figure 4.10.

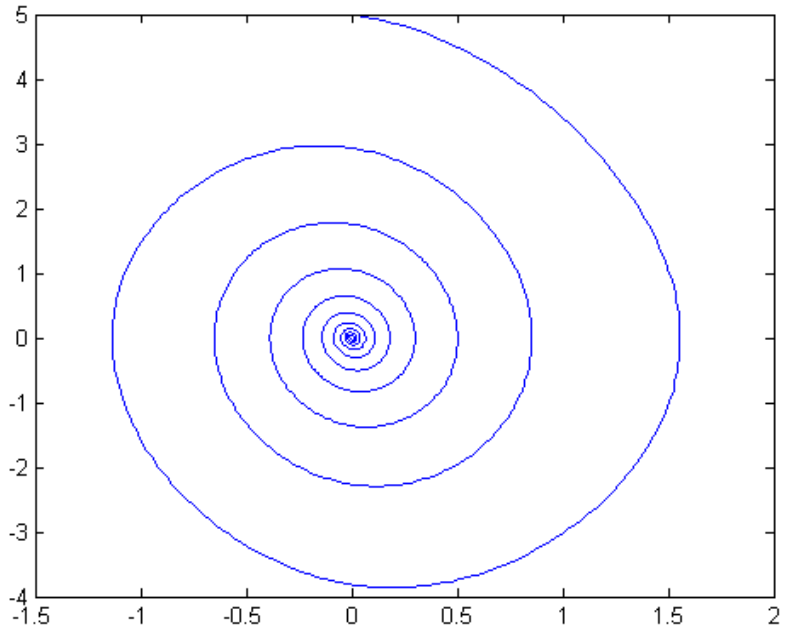

Figure 4.10: The pendulum motion (first scenario).

In the next scenario we increase the initial velocity to 10.
Then the change in the MATLAB code is the following.

```
[t, xa] = ode45(g, [0:0.01:20], [0 10]);
plot(xa(:,1), xa(:,2))
```

This time the angle increases to over 14 radians before the curve spirals in to a point near (12.5, 0). More precisely, it spirals toward $(4\pi, 0)$, because 4π radians represents the same position for the pendulum as 0 radians does. The pendulum has swung overhead and made two complete revolutions before beginning its damped oscillation toward its rest position. The velocity at first decreases but then rises after the angle passes through π, as the pendulum passes the upright position and gains momentum. The pendulum has just enough momentum to swing through the upright position once more at the angle 3π. The corresponding picture is shown in Figure 4.11.

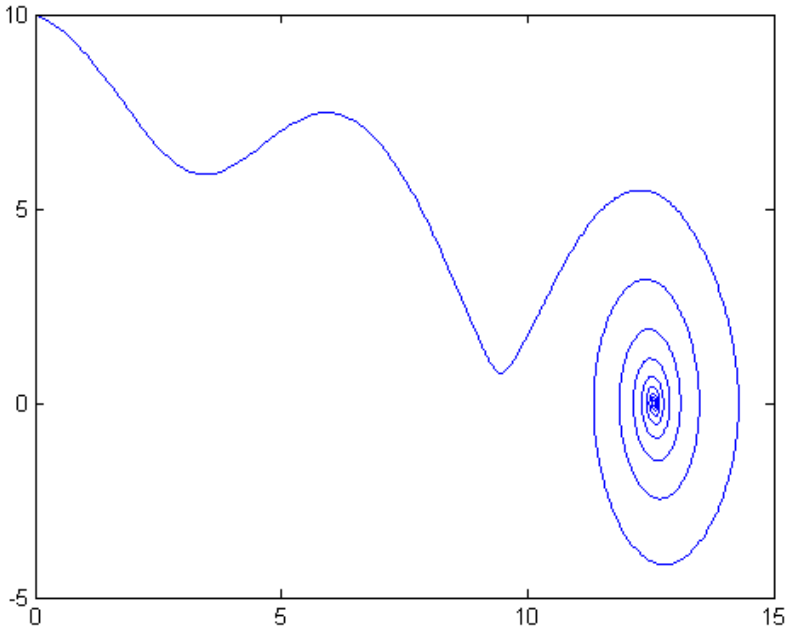

Figure 4.11: The pendulum motion (second scenario).

In the sequel our aim is finding the initial angular velocity that causes the pendulum to swing overhead.

Suppose that we want to find, to within 0.1, the minimum initial velocity required to make the pendulum, starting from its rest position, swing overhead once. It will be useful to be able to see the solutions corresponding to several different initial velocities on one graph. First we consider the integer velocities 5 to 10.

The corresponding MATLAB code can be written as follows.

```
% MATLAB code for finding the initial angular
% velocity that causes the pendulum to swing overhead.
% We use the built-in function ODE45
hold on
for a = 5:10 [t, xa] = ode45(g, [0:0.01:20], [0 a]);
plot(xa(:, 1), xa(:, 2))
end
hold off
```

The corresponding picture is shown in Figure 4.12.

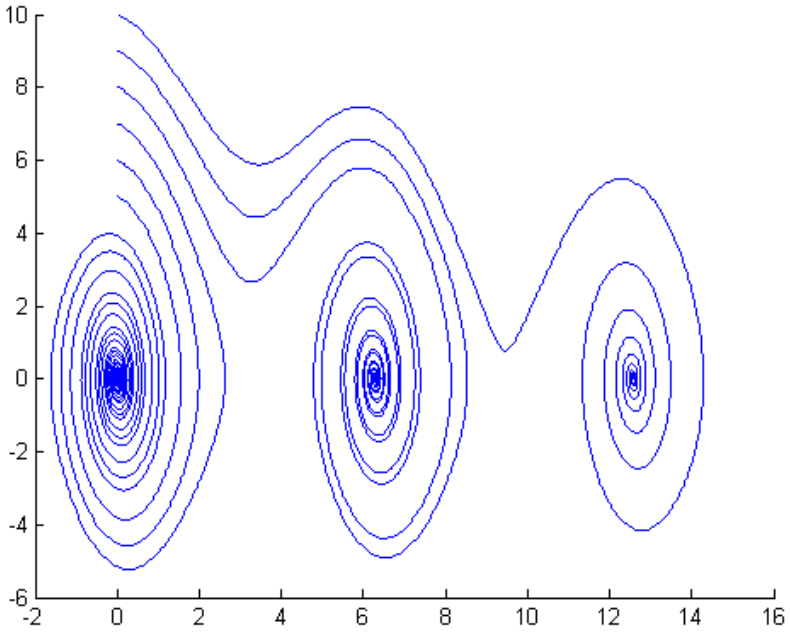

Figure 4.12: The pendulum motion for initial velocities 5 to 10.

Initial velocities $5, 6$, and 7 are not large enough for the angle to increase past π, but initial velocities $8, 9$, and 10 are enough to make the pendulum swing overhead.

Let us see what happens between 7 and 8. To this aim, we run the MATLAB code as follows.

```
hold on
for a = 7.0:0.2:8.0 [t, xa] = ode45(g, [0:0.01:20], [0 a]);
plot(xa(:, 1), xa(:, 2))
end
hold off
```

The corresponding picture is shown in Figure 4.13.

We see that the cut-off is somewhere between 7.2 and 7.4. Let us make one more refinement.

```
hold on
for a = 7.2:0.05:7.4 [t, xa] = ode45(g, [0:0.01:20], [0 a]);
plot(xa(:, 1), xa(:, 2))
end
```

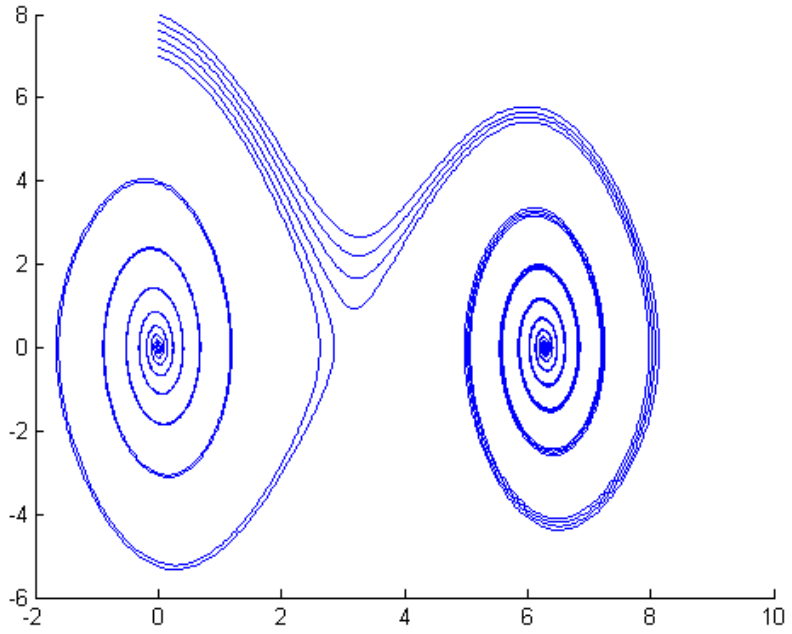

Figure 4.13: The pendulum motion for initial velocities between 7 and 8.

```
hold off
```

The corresponding picture is shown in Figure 4.14.

Hence we conclude that the minimum velocity needed is somewhere between 7.25 and 7.3.

4.3.2 Numerical solution of the heat conduction equation

In this section we will use MATLAB to numerically solve the heat conduction equation (also known as the diffusion equation), a partial differential equation that describes many physical processes such as conductive heat flow or the diffusion of an impurity in a motionless fluid. We can picture the process of diffusion as a drop of dye spreading in a glass of water. (To a certain extent we could also picture cream in a cup of coffee, but in that case the mixing is generally complicated by the fluid motion caused by pouring the cream into the coffee, and is further accelerated by stirring the coffee.) The dye consists of a large number of individual particles, each of which repeatedly bounces off the

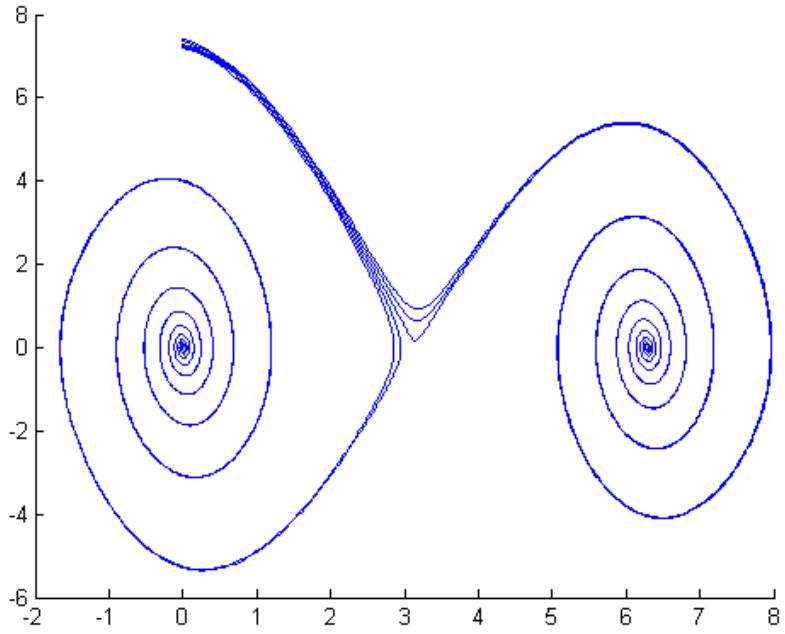

Figure 4.14: The pendulum motion for initial velocities between 7.2 and 7.4.

surrounding water molecules, following an essentially random path. There are so many dye particles that their individual random motions form an essentially deterministic overall pattern as the dye spreads evenly in all directions (we ignore here the possible effect of gravity). In a similar way, we can imagine heat energy spreading through random interactions of nearby particles.

In a homogeneous three-dimensional medium, the heat conduction equation can be given as

$$\frac{\partial u}{\partial t} = k\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right).$$

Here u is a function of t, x, y, and z that represents the temperature, or concentration of impurity in the case of diffusion, at time t at position (x, y, z) in the medium. The constant k depends on the materials involved, and is called the thermal conductivity in the case of heat flow, and the diffusion coefficient in the case of diffusion. To simplify matters, let us assume that the medium is instead one-dimensional. This could represent diffusion in a thin water-filled tube or heat flow in a thin insulated wire; let us think primarily of the case of

heat flow. Then the partial differential equation becomes

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2},$$

where $u(x,t)$ is the temperature at time t a distance x along the wire.

To solve this partial differential equation we need both initial conditions of the form $u(x,0) = f(x)$, where $f(x)$ gives the temperature distribution in the wire at time $t = 0$, and boundary conditions at the endpoints of the wire, call them $x = a$ and $x = b$. We choose so-called Dirichlet boundary conditions $u(a,t) = Ta$ and $u(b,t) = Tb$, which correspond to the temperature being held steady at values Ta and Tb at the two endpoints. Though an exact solution is available in this scenario, let us instead illustrate the numerical method of finite differences.

To begin with, on the computer we can keep track of the temperature u only at a discrete set of times and a discrete set of positions x. Let the times be $0, \Delta t, 2\Delta t, \ldots, N\Delta t$, and let the positions be $a, a + \Delta x, a + J\Delta x = b$, and let $u_j^n = u(a + j\Delta x, n\Delta t)$. After rewriting the partial differential equation in terms of finite-difference approximations to the derivatives, we get

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = k\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}.$$

(These are the simplest approximations we can use for the derivatives, and this method can be refined by using more accurate approximations, especially for the t derivative.) Thus if, for a particular n, we know the values of u_j^n for all j, we can solve the equation above to find for each j,

$$\begin{aligned} u_j^{n+1} =& u_j^n + \frac{k\Delta t}{(\Delta x)^2}\left(u_{j+1}^n - 2u_j^n + u_{j-1}^n\right) \\ =& s\left(u_{j+1}^n + u_{j-1}^n\right) + (1-2s)u_j^n, \end{aligned}$$

where $s = \frac{k\Delta t}{(\Delta x)^2}$.

In other words, this equation tells us how to find the temperature distribution at time step $n+1$ given the temperature distribution at time step n. (At the endpoints $j = 0$ and $j = J$, this equation refers to temperatures outside the prescribed range for x, but at these points we will ignore the equation above and apply the boundary conditions instead.) We can interpret this equation as saying that the temperature at a given location at the next time step is a weighted average of its temperature and the temperatures of its neighbors at the current time step. In other words, in time Δt, a given section of the wire of length Δx transfers to each of its neighbors a portion s of its heat energy and

keeps the remaining portion $1-2s$. Thus our numerical implementation of the heat conduction equation is a discretized version of the microscopic description of diffusion we gave initially, that heat energy spreads due to random interactions between nearby particles. The following M-file, which we have named heat2013.m, iterates the procedure described above.

```
 function u = heat2013(k, t, x, init, bdry)
% Solve the 1D heat conduction equation on
%the rectangle described by vectors x and t
% with initial condition u(t(1), x) = init
%with Dirichlet boundary conditions
%u(t, x(1)) = bdry(1),
% and u(t, x(end)) = bdry(2).
 J = length(x); N = length(t);
 dx = mean(diff(x)); dt = mean(diff(t));
 s = k*dt/dx^2; u = zeros(N,J);
 u(1,:)  = init;
 for n = 1:N-1
 u(n+1,2:J-1) = s*(u(n,3:J) + u(n,1:J-2))+....
 (1-2*s)*u(n,2:J-1);
 u(n+1,1) = bdry(1);
 u(n+1,J) = bdry(2);
 end
 end
```

The function HEAT2013 takes as inputs the value of k, vectors of t and x values, a vector *init* of initial values (which is assumed to have the same length as x), and a vector *bdry* containing a pair of boundary values. Its output is a matrix of u values.

Remark 4.3.1. Since indices of arrays in MATLAB must start at 1, not 0, we have deviated slightly from our earlier notation by letting $n = 1$ represent the initial time and $j = 1$ represent the left endpoint.) We also note that, in the first line following the for statement, we compute an entire row of u, except for the first and last values, in one line; each term is a vector of length $J-2$, with the index j increased by 1 in the term $u(n, 3:J)$ and decreased by 1 in the term $u(n, 1:J-2)$. In programming this method is called vectorization.

Let us use the M-file above to solve the one-dimensional heat conduction equation with $k = 2$ on the interval $-5 \leq x \leq 5$ from time $t = 0$ to time $t = 4$, using boundary temperatures 15 and 25, and an initial temperature distribution of 15 for $x < 0$ and 25 for $x > 0$. One can imagine that two separate wires of length 5 with different temperatures are joined at time $t = 0$

at position $x = 0$, and each of their far ends remains in an environment that holds it at its initial temperature. We must choose values for Δt and Δx; let us try $\Delta t = 0.1$ and $\Delta x = 0.5$, so that there are 41 values of t ranging from 0 to 4 and 21 values of x, ranging from -5 to 5.

Then the MATLAB code for this case can be written as follows.

```
tvals = linspace(0, 4, 41);
xvals = linspace(-5, 5, 21);
init = 20 + 5*sign(xvals);
uvals = heat2013(2, tvals, xvals, init, [15 25]);
surf(xvals, tvals, uvals); xlabel x;
ylabel t; zlabel u
```

The corresponding picture is shown in Figure 4.15.

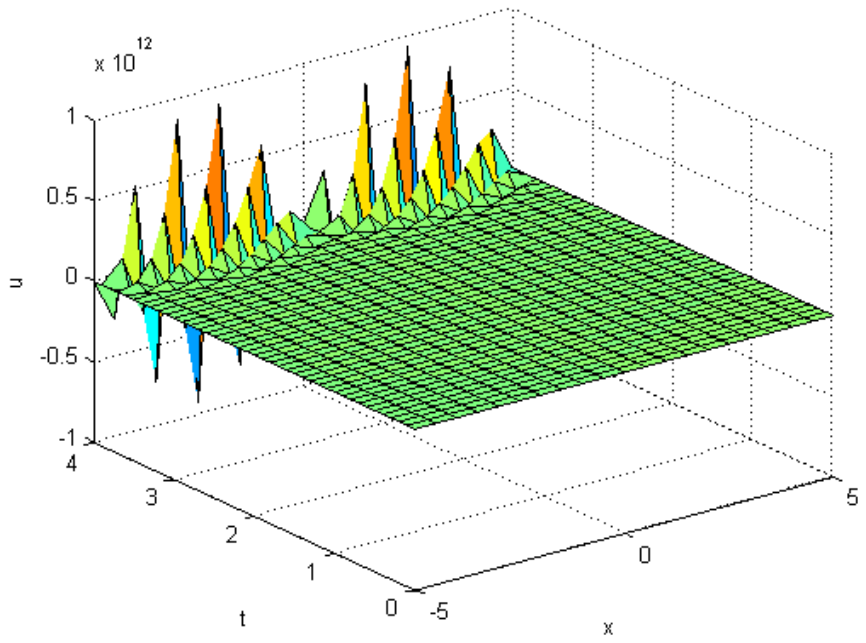

Figure 4.15: The numerical solution of the heat conduction equation by using $k = 2$, $\Delta x = 0.5$, and $\Delta t = 0.1$.

Here we used surf to show the entire solution $u(x, t)$. The output is clearly unrealistic; notice the scale on the u axis! The numerical solution of partial differential equations is fraught with dangers, and instability like that seen above is a common problem with finite difference schemes. For many partial

differential equations a finite difference scheme will not work at all, but for the heat conduction equation and similar equations it will work well with proper choice of Δt and Δx. One might be inclined to think that since our choice of Δx was larger, it should be reduced, but in fact this would only make matters worse. Ultimately the only parameter in the iteration we are using is the constant s, and one drawback of doing all the computations in an M-file as we did above is that we do not automatically see the intermediate quantities it computes. In this case we can easily calculate that $s = 2(0.1)/(0.5)2 = 0.8$. Notice that this implies that the coefficient $1 - 2s$ of u_j^n in the iteration above is negative. Thus the "weighted average" we described before in our interpretation of the iterative step is not a true average; each section of wire is transferring more energy than it has at each time step! The solution to the problem above is thus to reduce the time step Δt; for instance, if we cut it in half, then $s = 0.4$, and all coefficients in the iteration are positive.

Hence, the corresponding MATLAB code is the following.

```
tvals = linspace(0, 4, 81);
xvals = linspace(-5, 5, 21);
init = 20 + 5*sign(xvals);
uvals = heat2013(2, tvals, xvals, init, [15 25]);
surf(xvals, tvals, uvals); xlabel x;
ylabel t; zlabel u
```

The corresponding picture is shown in Figure 4.16.

This looks much better! As time increases, the temperature distribution seems to approach a linear function of x. Indeed $u(x, t) = 20 + x$ is the limiting "steady state" for this problem; it satisfies the boundary conditions and it yields 0 on both sides of the partial differential equation.

Generally speaking, it is best to understand some of the theory of partial differential equations before attempting a numerical solution like we have done here. However, for this particular case at least, the simple rule of thumb of keeping the coefficients of the iteration positive yields realistic results. A theoretical examination of the stability of this finite difference scheme for the one-dimensional heat conduction equation shows that indeed any value of s between 0 and 0.5 will work, and it suggests that the best value of Δt to use for a given Δx is the one that makes $s = 0.25$.

Remark 4.3.2. Notice that while we can get more accurate results in this case by reducing Δx, if we reduce it by a factor of 10 we must reduce Δt by a factor of 100 to compensate, making the computation take 1000 times as long and use 1000 times the memory!(In case, when we store the results of each time stepping.)

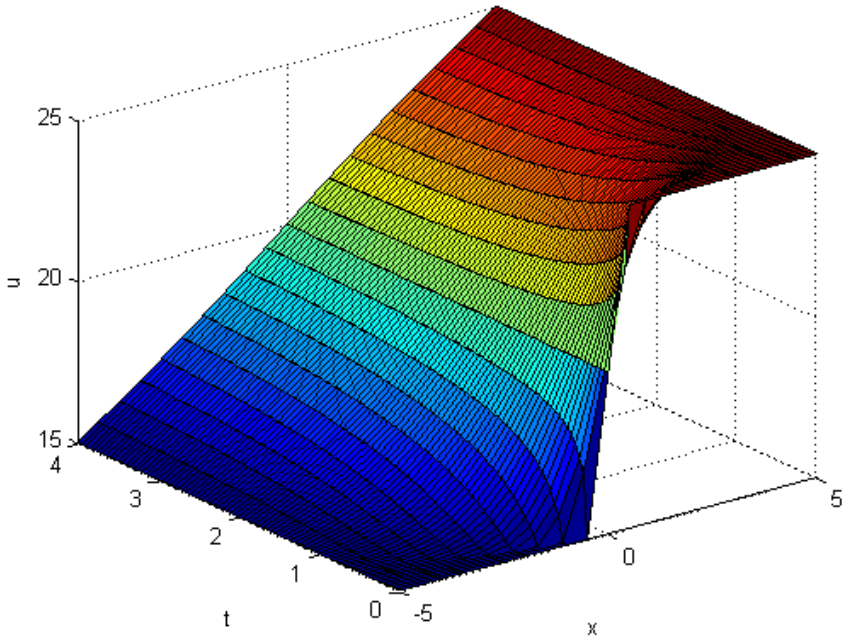

Figure 4.16: The numerical solution of the heat conduction equation by using $k = 2$, $\Delta x = 0.5$, and $\Delta t = 0.05$.

In the sequel we consider a more general class of the heat equation. Earlier we mentioned that the problem we solved numerically could also be solved analytically. The value of the numerical method is that it can be applied to similar partial differential equations for which an exact solution is not possible or at least not known. For example, consider the one-dimensional heat conduction equation with a variable coefficient, representing an inhomogeneous material with varying thermal conductivity $k(x)$,

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k(x)\frac{\partial u}{\partial x}\right) = k(x)\frac{\partial^2 u}{\partial x^2} + k'(x)\frac{\partial u}{\partial x}.$$

For the first derivatives on the right-hand side, we use a symmetric finite difference approximation, so that our discrete approximation to the partial differential equations becomes

$$\begin{aligned}\frac{u_j^{n+1} - u_j^n}{\Delta t} =& k_j \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} + \\ & \frac{k_{j+1} - k_{j-1}}{2\Delta x}\frac{u_{j+1} - u_{j-1}}{2\Delta x},\end{aligned}$$

where $k_j = k(a + j\Delta x)$. Then the time iteration for this method reads as

$$\begin{aligned} u_j^{n+1} =& s_j \left(u_{j+1}^n + u_{j-1}^n\right) + (1 - 2s_j)u_j^n + \\ & 0.25(s_{j+1} - s_{j-1}) \left(u_{j+1}^n - u_{j-1}^n\right), \end{aligned}$$

where $s_j = k_j \Delta t/(\Delta x)^2$. In the following M-file, which we called heat2013vc.m, we modify our previous M-file to incorporate this iteration.

```
function u = heat2013vc(k, t, x, init, bdry)
% Solve the 1D heat conduction equation
% with variable coefficient k on the rectangle
% described by vectors x and t
% with u(x, t(1)) = init initial and
% Dirichlet boundary conditions
% u(x(1), t) = bdry(1), u(x(end), t) = bdry(2).
 J = length(x); N = length(t);
 dx = mean(diff(x)); dt = mean(diff(t));
 s = k*dt/dx^2; u = zeros(N,J);
 u(1,:)  = init;
 for n = 1:N-1
 u(n+1, 2:J-1) = s(2:J-1).*(u(n, 3:J) +...
 (u(n, 1:J-2)) +1 - 2*s(2:J-1)).*...
 u(n,2:J-1) + ...
 0.25*(s(3:J) - s(1:J-2)).*(u(n, 3:J) -...
 u(n, 1:J-2));
 u(n+1,1) = bdry(1);
 u(n+1,J) = bdry(2);
 end
 end
```

Notice that k is now assumed to be a vector with the same length as x and that as a result so is s. This in turn requires that we use vectorized multiplication in the main iteration, which we have now split into three lines. Lets use this M-file to solve the one-dimensional variable-coefficient heat conduction equation with the same boundary and initial conditions as before, using $k(x) = 1 + (x/5)^2$. Since the maximum value of k is 2, we can use the same values of Δt and Δx as before.

Hence, the corresponding MATLAB code is the following.

```
 tvals = linspace(0, 4, 81);
 xvals = linspace(-5, 5, 21);
 init = 20 + 5 * sign(xvals);
 kvals = 1 + (xvals/5).^2;
```

```
uvals = heat2013vc(kvals, tvals, xvals,...
init, [15 25]);
surf(xvals, tvals, uvals); xlabel x;
ylabel t; zlabel u
```

The corresponding picture is shown in Figure 4.17.

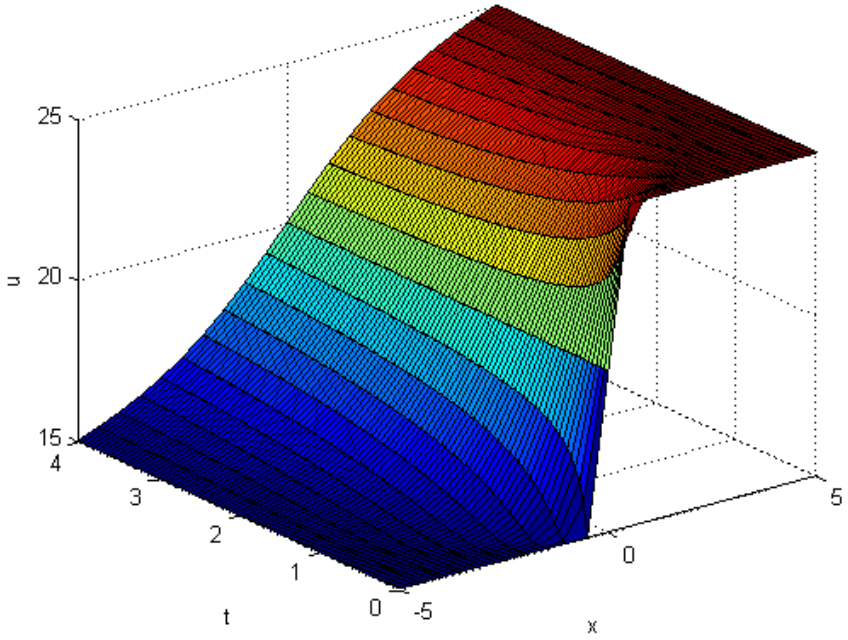

Figure 4.17: The numerical solution of the heat conduction equation by using variable $k(x)$, $\Delta x = 0.5$, and $\Delta t = 0.05$.

In this case the limiting temperature distribution is not linear; it has a steeper temperature gradient in the middle, where the thermal conductivity is lower. Again one could find the exact form of this limiting distribution, $u(x,t) = 20(1 + (1/\pi)\arctan(x/5))$, by setting the t-derivative to zero in the original equation and solving the resulting ordinary differential equation. We can use the method of finite differences to solve the heat conduction equation in two or three space dimensions as well. For this and other partial differential equations with time and two space dimensions, we can also use the PDE Toolbox, which implements the more sophisticated finite element method.

Part II

Numerical methods for boundary value problems

Chapter 5

Motivation

We have shown that for the uniqueness of the solution of some ordinary differential equation it is necessary to give additional conditions. In the previous part these conditions were the initial conditions, which means that at some initial time point (which is usually $t = 0$) we give some property of the solution. Typically, for an ordinary differential equation of order m we specify the derivatives of order $k = 0, 1, \ldots, m-1$ of the solution at this starting point. (Such a problem is called Cauchy problem.) E.g., for the second order equation

$$u'' = f(t, u, u') \tag{5.1}$$

we impose the conditions

$$u(0) = u_0; \quad u'(0) = u_0', \tag{5.2}$$

where u_0, u_0' are given numbers. However, it is very typical that, investigating the equation (5.1) on the bounded and closed interval $[0, T]$, we know the values of the solution at both end-points of this interval, and not the derivative at the starting point. This means that for the equation (5.1) we prescribe the additional conditions

$$u(0) = u_0, \quad u(T) = u_1. \tag{5.3}$$

In the following we give an example which leads to a problem of the form (5.1)-(5.3).

Example 5.0.1. *We launch a cannonball from a fixed place, called origin. Let $y(t)$ denote the vertical distance of the cannonball at $t > 0$, and $x(t)$ the horizontal distance from the origin. We assume that*

- *the horizontal velocity is constant;*
- *the vertical distance (height of the ball) depends only on the gravitation.*

Let us define the angle of the launch under which the cannonball lands at a prescribed distance, e.g., at the place $x = L$.

First we define the continuous mathematical model of the problem. We denote by $v > 0$ the constant horizontal velocity. Then the corresponding equations are

$$\begin{aligned} \dot{x}(t) &= v \\ \ddot{y}(t) &= -g. \end{aligned} \tag{5.4}$$

Clearly, $x(0) = 0$ and $y(0) = 0$. (Since at the starting moment $t = 0$ the cannonball is at the origin, hence the corresponding distances are zero.) The solution of the first equation, taking into account the initial condition, is $x(t) = vt$, i.e., $t = x/v$. We introduce the new function as $y(t) = y(x/v) =: Y(x)$. Then, using the chain rule of differentiation, we get

$$\begin{aligned} \dot{y}(t) &= \frac{dY}{dx}\frac{dx}{dt} = \frac{dY}{dx}v, \\ \ddot{y}(t) &= v\frac{d^2Y}{dx^2}v = v^2\frac{d^2Y}{dx^2}. \end{aligned}$$

For the new function the following relations hold:

$$\begin{aligned} Y''(x) = -\frac{g}{v^2}, \quad x \in (0, L) \\ Y(0) = 0, \;\; Y(L) = 0. \end{aligned} \tag{5.5}$$

Since the differential equation can be integrated easily, the solution of problem (5.5) can be defined directly:

$$Y(x) = \frac{gx}{2v^2}(L - x).$$

Hence, the unknown angle α will be determined from the formula

$$\mathrm{tg}(\alpha) = Y'(0) = \frac{gL}{2v^2}. \tag{5.6}$$

Remark 5.0.3. Formula (5.6) says that for any distance (i.e., for an arbitrary L) there exists some suitable angle, which is obviously impossible. The reason for this contradiction is that our model is too simple: for the long distance the physical assumptions made are not realistic, and, additionally, we should involve some other important physical factors, too.

The change of the variable t to x is motivated by the reason that the boundary value problem(5.1)-(5.3) is considered on some bounded *spatial domain*. Therefore, in the sequel we will prefer this notation.

Definition 5.0.2. *The problem*

$$\begin{aligned} u'' =& f(x,u,u'), \quad x \in (a,b), \\ & u(a)=\alpha, \quad u(b)=\beta \end{aligned} \tag{5.7}$$

for the unknown function $u = u(x)$ is called two-point boundary value problem.

We always assume that the problem (5.7) has a unique solution.

There are several theorems which guarantee this condition. A useful statement is the following.

Theorem 5.0.3. *Let $T = \{(x,\ p_1,\ p_2) : x \in [a;b],\ p_1,\ p_2 \in \mathbb{R}\} \subset \mathbb{R}^3$ and $f : T \to R$ a given function. (I.e., $f = f(x,p_1,p_2)$.) We assume that*

1. *$f \in C(T)$;*
2. *$\partial_1 f, \partial_2 f \in C(T)$;*
3. *$\partial_2 f > 0$ on T;*
4. *there exists a nonnegative constant M such that $|\partial_3 f| < M$ on T.*

Under these conditions the two-point boundary value problem (5.7) *has a unique solution.*

An important consequence of Theorem 5.0.3 is the following statement.

Corollary 5.0.4. Assume that f is a linear non-homogeneous function w.r.t. its second and third variable, i.e., we consider the boundary value problem of the form

$$\begin{aligned} u'' =& f(x,u,u') \equiv p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a)=\alpha, \quad u(b)=\beta, \end{aligned} \tag{5.8}$$

where $p, q, r \in C[a,b]$ are given continuous functions. If $q(x) > 0$ on $[a,b]$, then the linear boundary value problem (5.8) has a unique solution.

Remark 5.0.4. For the initial value problem we have seen that the existence of a unique solution depends on the smoothness of the right side of the differential equation. (Roughly speaking, when the right side is "good", then there exists a unique solution for any additional (i.e., initial) condition.) The following example shows that the situation is different for the boundary value problem.

Example 5.0.5. *Let us consider the equation* $u''+u=1$*, for which the general solution is obviously* $u(x)=C_1\cos x+C_2\sin x$*, where* C_1 *and* C_2 *are arbitrary constants. When the two boundary conditions are* $u(\pi/4)=2$ *and* $u(\pi)=2$*, then these constants can be defined uniquely, and the solution is* $u(x)=-\cos x+(\sqrt{2}+1)\sin x$*. However, for the conditions* $u(\pi/4)=2$ *and* $u(5\pi/4)=2$ *the constants cannot be defined, hence the problem has no solution.*

Remark 5.0.5. The above Example 5.0.5 shows that in Corollary 5.0.4 for the linear boundary value problem the condition $q(x)>0$ cannot be omitted, since in this example $q(x)=-1$, but all the other conditions for the function $f(t,p_1,p_2)=-p_1$ are satisfied.

We note that we can also give examples where there exists a solution of the boundary value problem, but it is not unique.

Example 5.0.6. *We consider the problem*

$$\begin{aligned} u'' &= -\exp(u+1), \quad x\in(0,1),\\ u(0) &= 0, \quad u(1)=0. \end{aligned} \tag{5.9}$$

This problem has the solution

$$u(x)=-2\ln\frac{\cosh[(x-0.5)\theta/2]}{\cosh(\theta/4)},$$

where θ *is the solution of the equation* $\theta=\sqrt{2e}\cos(\theta/4)$*. Since the latter algebraic equation has two solutions, therefore the problem* (5.9) *has also two different solutions.*

Usually, the exact solution of the boundary value problems is impossible, or too difficult, so we have to apply numerical methods. In the following we consider two basic classes of numerical methods for solving boundary value problems, namely, the shooting method and the finite difference method.

Chapter 6

Shooting method

We consider a second order ordinary differential equation with two boundary conditions (5.7). In this problem a, b, α and β are given constants, u is the unknown function with the variable $x \in [a, b]$, and f is a given function that specifies the differential equation. This is a two-point boundary value problem. An initial value problem, which was considered in the first part of the book, would require that the two conditions be given at the same value of x. For example, $u(a) = \alpha$, and $u'(a) = \beta$. Because of the two separate boundary conditions, the above two-point boundary value problem is more difficult to solve.

6.1 Description of the shooting method

The basic idea of the shooting method is to replace the above boundary value problem by some initial value problem. But of course we do not know the derivative of u at $x = a$. However, we can guess and then further improve the guess iteratively. More precisely, we treat $u'(a)$ as the unknown, and use the secant method or Newton's method (or other methods for solving nonlinear equations) to determine $u'(a)$.

First we describe the basic idea of the shooting method in general.

Previously we have shown that a higher order equation can be described as a first order system. (C.f. formulas (3.36)-(3.39).) The equation in the boundary value problem (5.7) can also be re-written in the form of a first order system of two unknown functions. Namely, we introduce the new function $\mathbf{u} : [a, b] \to \mathbb{R}^2$, $\mathbf{u}(x) = (u_1(x), u_2(x))$ as follows: $u_1(x) = u(x)$ and $u_2(x) = u'(x)$. Then we get the problem

$$\begin{aligned} &u_1' = u_2, \quad u_2' = f(x, u_1, u_2), \\ &u_1(a) = \alpha, \quad u_1(b) = \beta. \end{aligned} \tag{6.1}$$

Obviously, (6.1) cannot be solved, since we have additional conditions for u_1 only.

Therefore, we consider the first order system

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(t, \mathbf{u}), \quad t \in [a, b] \\ u_1(a) &= \alpha, \quad u_1(b) = \beta. \end{aligned} \tag{6.2}$$

The basic idea of the *shooting method* is that instead of problem (6.2) we consider the initial value problem

$$\begin{aligned} \mathbf{u}' &= \mathbf{f}(x, \mathbf{u}), \quad x \in [a, b] \\ \mathbf{u}(a) &= \boldsymbol{\alpha}, \end{aligned} \tag{6.3}$$

and our aim is to solve it. Since

$$\mathbf{u}(a) = \begin{pmatrix} u_1(a) \\ u_2(a) \end{pmatrix},$$

therefore, for the vector $\boldsymbol{\alpha} \in \mathbb{R}^2$, which defines the initial condition at $x = a$, we know only the first coordinate from the boundary condition (6.2). However, the second coordinate ($u_2(a) = u'(a)$) is unknown. Let us introduce the notation $s = u_2(a)$. The shooting method is based on the idea that in the vector

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha \\ s \end{pmatrix} \tag{6.4}$$

the value of s is chosen in such a way that the solution of (6.3) at the point $x = b$ would be equal to the prescribed value β. This means the following. We denote by $\mathbf{w}(x; s)$ the solution of the problem (6.2).[1] Our task is choosing a value $\tilde{s}$ such that the equality

$$w_1(b; \tilde{s}) = \beta \tag{6.5}$$

is satisfied. To this aim, we have to solve the (typically nonlinear) equation

$$g(s) = w_1(b; s) - \beta = 0. \tag{6.6}$$

In the sequel we give the exact description of the shooting method. We introduce a function w, which is a function of x, but it also depends on a parameter s. Namely, $w = w(x; s)$. We use w' and w'' to denote the partial

[1] The solution depends on the parameter s, and it is denoted in this way.

derivatives of w with respect to x. We want w to be exactly u, if s is properly chosen. But w is defined for any s, by the problem

$$\begin{aligned} w'' =& f(x, w, w'), \quad x \in (a, b), \\ & w(a; s) = \alpha, \quad w'(a; s) = s. \end{aligned} \tag{6.7}$$

If we choose some fixed s, then we can solve the above initial value problem for the unknown function $w(x; s)$, which is now a function of one variable x. In general $w(x; s)$ is not the same as $u(x)$, since $w'(a; s) = s \neq u'(a)$. But if $s = \tilde{s}$ is chosen in the way that $w'(a; \tilde{s}) = u'(a) = \tilde{s}$, then these two functions are equal. Since we donot know $u'(a)$, we determine it from the boundary condition at $x = b$. Namely, we define $\tilde{s}$ by solving the equation

$$g(s) = w(b, s) - \beta = 0 \tag{6.8}$$

for theunknown s. Denoting the solution by $\tilde{s}$, for this value we clearly have $w(b, \tilde{s}) = \beta$, and hence for the function $w(x; \tilde{s})$ of the variable x we have

$$\begin{aligned} w''(x, \tilde{s}) =& f(x, w(x, \tilde{s}), w'(x, \tilde{s})), \quad x \in (a, b), \\ & w(a; \tilde{s}) = \alpha, \quad w(b, \tilde{s}) = \beta. \end{aligned} \tag{6.9}$$

Due to the uniqueness of the solution of problem (5.7) this implies that $w(x; \tilde{s}) = u(x)$ for each $x \in [a, b]$.

6.2 Implementation of the shooting method

In the previous section we gave the description of the shooting method, more precisely, how to lead the boundary value problem to the solution of some initial value problem. We mention that if we can solve the initial value problem of w (for arbitrary s) analytically, (which is not typical) we can write down a formula for $g(s) = w(b, s) - \beta$. Of course, this is not possible in general. However, without an analytic formula, we can still solve $g(s) = 0$ numerically. For any s, a numerical method for the initial value problem of w can be used to find an approximate value of $w(b; s)$ (and thus $g(s)$).

Hence we can see that the solution of the initial value problem gives rise to some new difficulties.

1. We have to solve the sequence of auxiliary initial value problems (6.7). To this aim, we have to choose some suitable numerical method.

2. We have to solve some non-linear equation (6.8), which also requires the application of a numerical method.

The choice of the appropriate numerical method for solving the initial value problems can be done by using the results of the first part of the book. Therefore, in the sequel, we analyze only the second problem.

6.2.1 Method of halving the interval

One of the simplest method is the well-known *method of halving the interval*. This method requires to find the values s_1 and s_2 such that $g(s_1)g(s_2) < 0$, and then the root on the interval (c_1, c_2) can be defined by successive halving of the interval.

We can summarize the algorithm of the shooting method combined with the halving of the interval method.

1. We fix a value s.
2. With the initial value vector (6.4) we solve the initial value problem (6.2) on the interval $[a, b]$, applying some numerical method. (E.g. some Runge–Kutta method.)
3. In step 1 we find two different values s_1 and s_2 such that the solutions, defined in step 2 with these parameters have different sign at the point $x = b$.
4. With the value $s = 0.5(s_1 + s_2)$ we repeat step 2.
5. According to the sign of the solution with this parameter at the point $x = b$ we re-define the value s_1 or s_2, and we continue the process.
6. When $|g(s_k)|$ is less than some a'priori given number $\varepsilon > 0$, we terminate the iteration.

We list some properties of the method of halving the interval.

- From the construction it is obvious that the method is always convergent in first order to some root on the interval $[s_1, s_2]$.
- The convergence of the sequence to the root is not monotonic. (This is clear from the geometry of the method.)
- For the error of the method, arising after the n-th step, we have the estimation
$$e_n = |\tilde{s} - s_n| \leq \frac{s_2 - s_1}{2^{n+1}}, \tag{6.10}$$
which follows from the construction directly.

- Based on the error estimate (6.10) we can see that having

$$n_{max} = \frac{\ln(s_2 - s_1)/\varepsilon}{\ln 2} - 1$$

steps, the error $e_{n_{max}}$ is less than ε.

- In the above algorithm the stopping criterion $|g(s_n)| < \varepsilon$ can be replaced by the condition

$$\frac{|s_n - s_{n-1}|}{|s_{n-1}|} < \varepsilon.$$

- For the computation of the halving point x of some interval $[A, B]$, the formula $x = A + (B - A)/2$ is recommended, instead of the natural $x = (A+B)/2$. The reason is the computer interpretation of the number: for the second formula it might happen that $x \notin [A, B]$, that is, we go out from the original interval where the existence of the root is guaranteed. However, for the first formula we always have $x \in [A, B]$.

6.2.2 Secant method

For solving a nonlinear equation we can also use the secant method. Then for solving the equation for finding $\tilde{s}$, we construct the iteration

$$s_{j+1} = s_j - \frac{s_j - s_{j-1}}{g(s_j) - g(s_{j-1})} g(s_j), \quad j = 1, 2, \ldots \tag{6.11}$$

The algorithm (6.11) is a two-step (three-level) iteration.

Remark 6.2.1. For the secant method we need two initial guesses, s_0 and and s_1. When we know that the equation (6.8) has a root on the interval $[a, b]$, then we put $s_0 = a$ and $s_1 = b$. However, the iteration can also be divergent.

We can give conditions under which the iteration is convergent, moreover, we can characterize the rate of the convergence, too. Namely, the following statement is true.

Theorem 6.2.1. *Assume that g has a root $\tilde{s}$ on (a, b), it is twice continuously differentiable, and the first and the second derivatives do not take the zero value on $[a, b]$. Let m_1, M_2 be constants with the property $0 < m_1 \leq |g'(x)|$ and $0 < |g''(x)| \leq M_2$. If $\max\{|a - \tilde{s}|, |b - \tilde{s}|\} < 2m_1/M_2$, then the sequence produced by the secant method* (6.11) *is monotonically convergent to $\tilde{s}$, and the order of the convergence is $p = (1 + \sqrt{5})/2 \approx 1.618$.*

According to Theorem 6.2.1, we should define a sufficiently small interval $[a, b]$ on which there exists a solution. This can be obtained by the method of halving the interval.

We can summarize the algorithm of the shooting method combined with the secant method as follows.

1. We put $s_0 = a$ and $s_1 = b$.

2. By solving the problems

$$\begin{aligned} w_j''(x, s_j) =& f(x, w_j(x, s_j), w_j'(x, s_j)), \quad x \in [a, b], \\ & w(a; s_j) = \alpha, \quad w'(a; s_j) = s_j \end{aligned} \tag{6.12}$$

   for $j = 0, 1$ with some numerical method, at the mesh-points we define the approximations to the functions $w_0(x, s_0)$ and $w_1(x, s_0)$, respectively.

3. Using the values $w_0(b, s_0)$ and $w_1(b, s_0)$, defined numerically in the previous step, we compute $g(s_0) = w_0(b, s_0) - \beta$ and $g(s_1) = w_0(b, s_1) - \beta$.

4. Using the already known values s_0,s_1, $g(s_0)$ and $g(s_1)$, by the formula

$$s_2 = s_1 - \frac{s_1 - s_0}{g(s_1) - g(s_0)} g(s_1),$$

   which is obtained from (6.11) for $j = 1$, we compute s_2.

5. With the help of the value s_2 we define $w_2(x, s_2)$ as the numerical solution of the initial value problem

$$\begin{aligned} w_2''(x, s_j) =& f(x, w_2(x, s_2), w_2'(x, s_2)), \quad x \in [a, b], \\ & w(a; s_2) = \alpha, \quad w'(a; s_2) = s_2. \end{aligned} \tag{6.13}$$

6. We define $g(s_2) = w_2(b, s_2) - \beta$.

7. We define

$$s_3 = s_2 - \frac{s_2 - s_1}{g(s_2) - g(s_1)} g(s_2).$$

8. etc.

9. When $|g(s_k)|$ is less than some a'priori given number $\varepsilon > 0$, we terminate the iteration.

Remark 6.2.2. We note that in the algorithm of the secant method it is not recommended to choice a too small ε. The reason is that in this case in the denominator of the iteration step (6.11) there might be a cancellation of the exact digits, and, since the numbers $g(s_{i-1})$ and $g(s_{i-2})$ are very close to each other, we should divide with an extremely small number. This fact may result in numerical instability of the computational process. To avoid this difficulty, the choice of a bigger ε is suggested, and after the last step of the algorithm one should execute one more *interpolation step*. This can be realized in the following way. We prepare the table

$$\begin{array}{|c|c|c|c|} \hline g(s_0) & g(s_1) & \dots & g(s_n) \\ \hline s_0 & s_1 & \dots & s_n \\ \hline \end{array} \tag{6.14}$$

and define the interpolation polynomial of order n, based on these points. This means that we define polynomial $p(x)$ of order n such that the relations $p(s_i) = s_i$ are satisfied for any $i = 0, 1, \dots n$. Hence p is the interpolation polynomial to the inverse function of g (i.e., g^{-1}). Then the accepted approximation s_{n+1} to the root of the function g (i.e., an approximation to $\tilde{s}$) will be computed by the formula $p(0) = s_{n+1}$.

6.2.3 Newton's method

Another possibility to solve the non-linear equation (6.8) is Newton's method. For this method the unknown root $\tilde{s}$ of the equation is defined by the iteration

$$s_{j+1} = s_j - \frac{g(s_j)}{g'(s_j)}, \quad j = 0, 1, 2, \dots \tag{6.15}$$

In this iteration we need only one starting value (s_0), however, there is a new difficulty: at each step we have to know the derivative of the function g at the point s_j. This value can be defined in the following way.

We recall the relation (6.7) for the function $w(x; s)$:

$$\begin{aligned} w''(x,s) =& f(x, w(x,s), w'(x,s)), \quad x \in (a,b), \\ & w(a;s) = \alpha, \quad w'(a,s) = s. \end{aligned} \tag{6.16}$$

Differentiating these equation w.r.t. the parameter s (using again the chain rule), we get

$$\begin{aligned} \frac{\partial}{\partial s} w''(x,s) =& \partial_2 f(x, w(x,s), w'(x,s)) \frac{\partial w}{\partial s}(x;s) + \\ & \partial_3 f(x, w(x,s), w'(x,s)) \frac{\partial w'}{\partial s}(x;s), \quad x \in (a,b), \\ & \frac{\partial w}{\partial s}(a;s) = 0, \quad \frac{\partial w'}{\partial s}(a,s) = 1. \end{aligned} \tag{6.17}$$

Let us introduce the new function

$$v(x;s) = \frac{\partial w}{\partial s}(x;s).$$

Then, based on the equations in (6.17), we obtain

$$\begin{aligned} v''(x,s) =& \partial_2 f(x, w(x,s), w'(x,s))v(x;s) + \\ & \partial_3 f(x, w(x,s), w'(x,s))v'(x;s), \quad x \in (a,b), \\ & v(a;s) = 0, \quad v'(a,s) = 1. \end{aligned} \tag{6.18}$$

Since $g(s) = w(b,s) - \beta$, therefore

$$g'(s) = \frac{\partial w(b,s)}{\partial s} = v(b;s).$$

This means that knowing the values v at the point $x = b$ we can define $g'(s_j)$ in the Newton iteration (6.15).

But how to define v on the interval $[a,b]$?

The problems (6.16) and (6.18) together yield a second order system for the unknown functions w and v, which can be re-written in the form of a first order system with four unknown functions. Namely, we introduce the vectors $\mathbf{z} = (z_1, z_2, z_3, z_4)^\top = (w, w', v, v')^\top$, $\boldsymbol{\alpha}(s) = (\alpha, s, 0, 1)^\top \in \mathbb{R}^4$, and the function $\mathbf{F} : \mathbb{R}^5 \to \mathbb{R}^4$ as

$$\mathbf{F}(x, \mathbf{z}) = \begin{pmatrix} z_2 \\ f(x, z_1, z_2) \\ z_4 \\ \partial_2 f(x, z_1, z_2) z_3 + \partial_3 f(x, z_1, z_2) z_4 \end{pmatrix},$$

where $x \in [a,b]$ and $\mathbf{z} \in \mathbb{R}^4$. Then for the unknown $\mathbf{z}(x,s)$ (i.e., for the functions $z_i(x;s)$ $(i = 1,2,3,4)$) we can write (6.16) and (6.18) in the form

$$\mathbf{z}'(x;s) = \mathbf{F}(x, \mathbf{z}(x;s)), \quad \mathbf{z}(a,s) = \boldsymbol{\alpha}(s), \tag{6.19}$$

which, for any fixed s, yields an initial value problemfor a first order system. Hence, choosing some fixed s and solving numerically the Cauchy problem (6.19) on the interval $[a,b]$, the value $z_3(b,a)$ gives us the value $g'(s)$.

Newton's method can be either divergent or convergent, it depends on the choice of the starting point of the iteration. We can give conditions under which the method is convergent, moreover, we can characterize the rate of the convergence as well. Namely, the following statement is true.

Theorem 6.2.2. *Assume that g has a root $\tilde{s}$ on (a, b), it is twice continuously differentiable, and the first and the second derivatives do not take the zero value on $[a, b]$. Let m_1, M_2 denote the constants defined in Theorem 6.2.1. If the starting value s_0 is chosen such that the inequality $|e_0| < \min\{|a - \tilde{s}|, |b - \tilde{s}|, 2m_1/M_2\}$ holds, then the sequence produced by Newton's method is monotonically convergent with second order rate of convergence to $\tilde{s}$.*

According to Theorem 6.2.2, we should define a sufficiently small interval $[a, b]$ on which there exists a solution. This can be obtained by the method of halving the interval.

Remark 6.2.3. We can guarantee the convergence independently of the choice of the starting value under some condition for the function g. Namely, we assume that g is a twice continuously differentiable function. When g' and g'' is nowhere zero and $g(s_0)g''(s_0) > 0$, then the statement of Theorem 6.2.2 remains valid.

We can summarize the algorithm of the shooting method combined with Newton's method as follows.

1. We give some value to s_0.
2. By solving the system of initial value problem (6.19) with some numerical method with this s_0, we define the approximations at the mesh-points to $z(x, s_0)$.
3. Using this approximation, we define the values $g(s_0) = z_1(b)$ and $g'(s_0) = z_3(b)$.
4. Using the Newton iteration (6.15) for $j = 0$, we compute s_1.
5. With this new s_1 we repeat the steps from 2, and define s_2.
6. etc.
7. When $|g(s_k)|$ is less than some a'priori given number $\varepsilon > 0$, we terminate the iteration.

6.3 The numerical solution of linear problems

In this part we consider the application of the shooting method to linear boundary value problems of the form

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a) = \alpha, \quad u(b) = \beta, \end{aligned} \tag{6.20}$$

introduced under (5.8). We show that for the case where f is chosen by a special way, namely, f has the form (6.20), then the numerical algorithm, given in (5.7), is simpler.

We assume that the problem (6.20) is solved by the shooting method, with the choice $c = c_1$ and $c = c_2$. We denote by $u_1(x) = u(x, c_1)$ and $u_2(x) = u(x, c_2)$ the solutions of the corresponding Cauchy problems, respectively. This means that the relations

$$\begin{aligned} u_1''(x) =& p(x)u_1'(x) + q(x)u_1(x) + r(x), \quad x \in [a,b], \\ & u_1(a) = \alpha, \quad u_1'(a) = c_1, \end{aligned} \tag{6.21}$$

$$\begin{aligned} u_2''(x) =& p(x)u_2'(x) + q(x)u_2(x) + r(x), \quad t \in [a,b], \\ & u_2(a) = \alpha, \quad u_2'(a) = c_2 \end{aligned} \tag{6.22}$$

are true. Let us introduce a new function

$$w(x) = \lambda u_1(x) + (1-\lambda)u_2(x), \tag{6.23}$$

where $\lambda \in \mathbb{R}$ is a fixed, at this moment arbitrary parameter. Since problem (6.20) is linear, therefore a convex linear combination of the solutions by (6.23) is also a solution of the problem, i.e.,

$$w''(x) = p(x)w'(x) + q(x)w(x) + r(x) \quad x \in [a,b]. \tag{6.24}$$

On the other hand, $w(a) = \alpha$ and $w(b) = \lambda u_1(b) + (1-\lambda)u_2(b)$. Therefore, we choose such a parameter λ under which the equality $w(b) = \beta$ is true, i.e., $\lambda u_1(b) + (1-\lambda)u_2(b) = \beta$. Hence we can define the value λ as

$$\lambda = \frac{\beta - u_2(b)}{u_1(b) - u_2(b)}. \tag{6.25}$$

Therefore, when λ is chosen by (6.25), then the function $w(x)$ in (6.23) is the solution of the linear boundary value problem (6.20).

The computer realization of this algorithm can be done as follows.

We consider the following two Cauchy problems:

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a) = \alpha, \quad u'(a) = 0, \end{aligned} \tag{6.26}$$

and

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a) = \alpha, \quad u'(a) = 1. \end{aligned} \tag{6.27}$$

We denote by $u_1(x)$ the solution of problem (6.26), and by $u_2(x)$ the solution of problem (6.27). (The problems (6.26) and (6.27) can be solved on the interval $[a,b]$ by re-writing them in the form of a first order system of two unknown functions, and applying any numerical method, discussed above.)

Then, using formula (6.25), we define the value of parameter λ. Finally, the approximated values of the solution of the linear boundary value problem (6.20) can be directly defined by use of the numerical approximations of $u_1(x)$ and $u_2(x)$ (which we have already defined) by the formula (6.23).

Remark 6.3.1. We can raise the following question: can we express λ by the formula (6.25) in any case? The following theorem gives the answer to this question [7].

Theorem 6.3.1. *Assume that the linear boundary value problem* (6.20) *has a unique solution. Then two cases are possible:*

1. *The function $u_1(x)$ from* (6.21) *is the solution of the problem* (6.20).
2. *The inequality $u_1(b) - u_2(b) \neq 0$ holds, and therefore λ can be defined by the formula* (6.25).

This means that for well-posed problems the numerical solution can always be defined.

We note that the solution of the well-posed linear boundary value problem (6.20) can be computed not only by a convex linear combination of the solutions of the Cauchy problems (6.26) and (6.27), but also by a combination of other Cauchy problems. To this aim, we consider the following two Cauchy problems:

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a) = \alpha, \quad u'(a) = 0, \end{aligned} \tag{6.28}$$

and

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a,b], \\ & u(a) = 0, \quad u'(a) = 1. \end{aligned} \tag{6.29}$$

We denote again by $u_1(x)$ the solution of problem (6.28), and by $u_2(x)$ the solution of problem (6.29). Then, as we can easily verify, the function

$$w(x) = u_1(x) + \frac{\beta - u_1(b)}{u_2(b)} u_2(x)$$

is a solution of problem (6.20). Hence, it is enough to solve numerically the Cauchy problems (6.28) and (6.29).

Let us summarize the results obtained for linear boundary value problems. As we have seen, for the numerical solution of these problems, in fact, the shooting method is not applied, since the solution can be represented as a sum of the solutions of two Cauchy problems for second order ordinary differential equations of a simpler structure. Compared to the general case, this results in a significant reduction of the computational costs: for linear problems we have to use some one-step numerical method for solving the initial value problems, while in the general case there is a need for numerical solution of the non-linear equation (6.8) at each iteration step.

Chapter 7

Finite difference method

As we saw in the last chapter, the use of the shooting method is demanding, an there are several open questions (such as the choice of the suitable numerical method for solving the initial value problem or the choice of the numerical method for solving the non-linear algebraic equations).

In this chapter, we describe the numerical solutions of two-point boundary value problems by using finite difference methods. The finite difference technique is based upon approximations that allow to replace the differential equations by finite difference equations. These finite difference approximations are in algebraic form, and the unknown solutions are related to grid points. Thus, the finite difference solution basically involves three steps:

1. We define the sequence of the meshes on the solution domain $[a; b]$.
2. We approximate the given differential equation by the system of difference equations that relates the solutions to grid points.
3. We solve the above algebraic system of the equations.

Comparing with the shooting method, we will see that the algorithm and the implementation are simpler for the finite difference method. However, the basic problem is the proof of the convergence of the method. For some special cases we will prove the convergence and determine its order, too.

The method and the theory of finite difference methods may lead us directly to the theory of the numerical solution of partial differential equations as well. However, it is out of the scope of this book.

7.1 Introduction to the finite difference method

The aim of this section is giving a general introduction to the finite difference method, and describe the first insight into the method. We mention that, in fact, this part is independent of Section 7.2, which gives a deeper theory of the method. For the easier understanding, in the latter chapters we will repeat the definitions of those notions which will be given in this section. Therefore, sections 7.2.1 and 7.2.2 can be followed without reading this introductory section.

7.1.1 Construction of the finite difference method

Consider the problem

$$\begin{aligned} -u'' + cu = f, \quad x \in (0,l), \\ u(0) = \alpha, \quad u(l) = \beta, \end{aligned} \tag{7.1}$$

where $c \geq 0$ is some constant, and f is a given continuous function. Since for general f the solution of this problem cannot be defined analytically, we construct some numerical method. The method consists of the following steps.

1. We define the sequence of uniform *meshes* ω_h on the interval $[0,l]$ as $\omega_h = \{x_i = ih,\ i = 1,2,\dots,N-1,\ h = l/N\ \}$ and $\overline{\omega}_h = \{x_i = ih,\ i = 0,1,\dots,N,\ h = l/N\ \}$. We use the notation $\gamma_h = \overline{\omega}_h \backslash \omega_h = \{x_0 = 0; x_N = l\}$ for the boundary points.

2. We use the notations $\mathbb{F}(\overline{\omega}_h)$ and $\mathbb{F}(\omega_h)$ for the vector space of the *mesh-functions* defined on the meshes $\overline{\omega}_h$ and ω_h, respectively, and mapping into $\mathbb{R}$.

3. Our aim is to find a *mesh-function* $y_h \in \mathbb{F}(\overline{\omega}_h)$ which is close to the solution of the function u – the solution of the problem (7.1) – at the mesh-points of $\overline{\omega}_h$, and, moreover, by refining the mesh (i.e., for $h \to 0$) their difference goes to zero.

The numerical method is defined by the principle (way) of definition of the mesh-function. The following idea seems to be quite natural. We consider the equation (7.1) at the mesh-points ω_h. Then we have the equality

$$-u''(x_i) + cu(x_i) = f(x_i) \tag{7.2}$$

for each $x_i \in \omega_h$. According to the usual finite difference approximation of the derivatives, we can use the following approximations for the derivatives at the point x_i:

$$u'(x_i) \approx \frac{u(x_i+h)-u(x_i)}{h}, \quad u'(x_i) \approx \frac{u(x_i)-u(x_i-h)}{h}. \tag{7.3}$$

Similarly, on the basis of these relations, for the second derivative we have the approximation

$$u''(x_i) \approx \frac{u(x_i+h)-2u(x_i)+u(x_i-h)}{h^2}. \tag{7.4}$$

According to our expectation, the mesh-function $y_h(x_i)$ approximates the solution of the boundary value problem at the mesh-points, i.e., $y_h(x_i) \approx u(x_i)$. Therefore, using the approximations (7.2) and (7.4) for the equation (7.2), we get

$$-\frac{y_h(x_i+h)-2y_h(x_i)+y_h(x_i-h)}{h^2} + cy_h(x_i) = f(x_i) \tag{7.5}$$

for each $x_i \in \omega_h$. Since at the boundary points the solution is known, therefore for these points we put

$$y_h(x_0) = \alpha, \quad y_h(x_N) = \beta. \tag{7.6}$$

The problem (7.5)-(7.6) is an equation for the unknown mesh-function $y_h \in \mathbb{F}$, and this task can be written in a compact form as follows. We denote by $L_h : \mathbb{F}(\overline{\omega}_h) \to \mathbb{F}(\overline{\omega}_h)$ the function which acts as follows: for an arbitrary mesh-function $v_h \in \mathbb{F}(\overline{\omega}_h)$ the mesh-function $L_h v_h \in \mathbb{F}(\overline{\omega}_h)$ is assigned by the formula

$$(L_h v_h)(x_i) = \begin{cases} -\dfrac{v_h(x_i+h)-2v_h(x_i)+v_h(x_i-h)}{h^2} + cv_h(x_i), & \text{when } x_i \in \omega_h; \\ v_h(x_i), & \text{when } x_i \in \gamma_h. \end{cases} \tag{7.7}$$

Let $b_h \in \mathbb{F}(\overline{\omega}_h)$ be the mesh-function, defined as

$$b_h(x_i) = \begin{cases} f(x_i), & \text{ha } x_i \in \omega_h; \\ \alpha, & \text{ha } x_i = x_0, \\ \beta, & \text{ha } x_i = x_N. \end{cases} \tag{7.8}$$

Using the notations (7.5)-(7.6) yield the following: find the mesh-function $y_h \in \mathbb{F}(\overline{\omega}_h)$ which is mapped by the operator L_h into the given mesh-function $b_h \in \mathbb{F}(\overline{\omega}_h)$, i.e., our task is the solution of the *operator equation*

$$L_h y_h = b_h. \tag{7.9}$$

We introduce the notations

$$y_h(x_i) = y_i,\ \ f(x_i) = f_i,\ \ \ b_h(x_i) = b_i.$$

Then the problem (7.5)-(7.6) can be written in the form

$$\begin{aligned} -\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + cy_i = f_i, \quad i = 1, 2, \ldots, N-1, \\ y_0 = \alpha, \quad y_N = \beta. \end{aligned} \tag{7.10}$$

Obviously, this is a system of linear algebraic equations of $N+1$ unknown values, which can be written in the usual "matrix form"

$$\mathbf{v}_h \mathbf{y}_h = \mathbf{b}_h, \tag{7.11}$$

where $\mathbf{y}_h = [y_0, y_1, \ldots, y_N]^T \in \mathbb{R}^{N+1}$ is unknown, while $\mathbf{b}_h = [\alpha, f_1, \ldots, f_{N-1}, \beta]^T \in \mathbb{R}^{N+1}$ is a given vector, and $\mathbf{v}_h \in \mathbb{R}^{(N+1)\times(N+1)}$ denotes the matrix

$$\mathbf{v}_h = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + c & -\frac{1}{h^2} & 0 & \ldots & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + c & -\frac{1}{h^2} & \ldots & 0 & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & 0 & -\frac{1}{h^2} & \frac{2}{h^2} + c & -\frac{1}{h^2} \\ 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.12}$$

7.1.2 Some properties of the finite difference scheme

The existence of a mesh-function $y_h \in \mathbb{F}$ defined by the operator equation (7.9) is equivalent to the solvability of the linear algebraic system (7.11). To this aim, we investigate the matrix $\mathbf{v}_h$.

M-matrices and their properties

In the sequel the key role is played by a special matrix class, called M-matrix. For the completeness of our investigation, we give the definition and the basic properties of such matrices. (For more details we refer to [2] and [14].)

Definition 7.1.1. *A non-singular matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with non-positive off-diagonal elements such that its inverse is non-negative*[1] *is called an M-matrix* [2].

[1] The ordering should be understood elementwise: a matrix (vector) is called non-negative when all its elements are non-negative. In notation we use "$\geq$". When each element is positive, then the matrix (vector) is called positive and we use the notation "$>$".

[2] The letter M in this definition refers to the first letter of the expression *(m)onotone matrix. This notion indicates such a property of the matrix that the relation $\boldsymbol{Ax} \geq \boldsymbol{Ay}$ implies the relation $\boldsymbol{x} \geq \boldsymbol{y}$.*

Example 7.1.2. *The matrix*

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

is an M-matrix, since the off-diagonal elements are non-positive, it is regular, and its inverse is

$$\boldsymbol{A}^{-1} = \begin{bmatrix} 3/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 3/4 \end{bmatrix},$$

which is non-negative.

Theorem 7.1.3. *The diagonal elements of an M-matrix are positive.*

Proof. The proof can be done indirectly. Assume that $\mathbf{A}$ is an M-matrix and there exists an index i such that $a_{ii} \leq 0$. Then for the i-th unit vector $\mathbf{e}_i = [0, 0, \ldots, 0, 1, 0, \ldots 0] \geq \mathbf{0}$ we get $\mathbf{A}\mathbf{e}_i \leq \mathbf{0}$. Hence, due to the property $\mathbf{A}^{-1} \geq \mathbf{0}$, we have $\mathbf{A}^{-1}\mathbf{A}\mathbf{e}_i = \mathbf{e}_i \leq \mathbf{0}$, which results in contradiction to the property $\mathbf{e}_i \geq \mathbf{0}$, because $\mathbf{e}_i \neq \mathbf{0}$. □

The conditions in the definition of the M-matrix cannot be checked easily, because they require the knowledge of the inverse matrix. Therefore it is very useful to know the following theorem, which gives the necessary and sufficient condition, and it can be checked much more easily.

Theorem 7.1.4. *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be a matrix with non-positive off-diagonal elements. Then $\boldsymbol{A}$ is an M-matrix if and only if there exists a positive vector $\boldsymbol{g} \in \mathbb{R}^n$ (i.e., $\boldsymbol{g} > \boldsymbol{0}$) such that $\boldsymbol{A}\boldsymbol{g} > \boldsymbol{0}$ is also positive (i.e., $\boldsymbol{A}\boldsymbol{g} > \boldsymbol{0}$).*

Proof. First we prove the necessity part of the theorem. Let $\mathbf{e} = [1, \ldots, 1]^T$. Then for the vector $\mathbf{g} = \mathbf{A}^{-1}\mathbf{e}$ we have the following properties. We know that $\mathbf{A}^{-1} \geq \mathbf{0}$, and it is regular, therefore each row consists of at least one non-zero (i.e., positive) element. Hence the sum of any row of $\mathbf{A}^{-1}$ is positive, therefore $\mathbf{g} > \mathbf{0}$. Having the relation $\mathbf{A}\mathbf{g} = \mathbf{A}\mathbf{A}^{-1}\mathbf{e} = \mathbf{e} > \mathbf{0}$, we have shown that $\mathbf{g}$ is a suitable choice in the statement.

For the sufficiency part we introduce the notations

$$\mathbf{G} = \operatorname{diag}(g_1, \ldots, g_n)$$

and

$$\mathbf{D} = \operatorname{diag}(a_{11}g_1, \ldots, a_{nn}g_n).$$

Since any element of $\mathbf{A}$ is positive (see Theorem 7.1.3), therefore $\mathbf{D}$ is regular. Then the matrix $\mathbf{D}^{-1}\mathbf{AG}$ has no positive element outside the main diagonal, and each element in its main diagonal is one. Hence we can write it in the form $\mathbf{D}^{-1}\mathbf{AG} = \mathbf{E} - \mathbf{B}$, where $\mathbf{B}$ is a non-negative matrix with zero entries in its main diagonal. Since $\mathbf{D}^{-1}\mathbf{AGe} = \mathbf{D}^{-1}\mathbf{Ag} > \mathbf{0}$, therefore $(\mathbf{E} - \mathbf{B})\mathbf{e} > \mathbf{0}$. This relation shows that the maximum norm of the matrix $\mathbf{B}$ less than 1. Consequently, for its spectral radius we have $\rho(\mathbf{B}) < 1$.[3], therefore the matrix $\mathbf{E} - \mathbf{B}$ is regular, and its inverse can be written in the form of a Neumann series.[4]

This implies the relation $\mathbf{0} \leq \mathbf{E} + \mathbf{B} + \mathbf{B}^2 + \cdots = (\mathbf{E} - \mathbf{B})^{-1}$. Consequently, $\mathbf{A}$ is also regular, ad its inverse has the form $\mathbf{A}^{-1} = \mathbf{G}(\mathbf{E} - \mathbf{B})^{-1}\mathbf{D}^{-1}$. Since each matrix on the right side of the product is non-negative, the statement of the theorem is proven. □

Corollary 7.1.6. This theorem proves that a matrix $\mathbf{A}$ with non-positive off-diagonal elements is an M-matrix if and only if there exists a vector $\mathbf{g} > \mathbf{0}$ such that $\mathbf{Ag} > \mathbf{0}$. Hence, this property can be considered as a definition of M-matrices, too. (For more equivalent definitions of M-matrices we refer to [2].)

The following theorem shows that using the above vectors $\mathbf{g}$ and $\mathbf{Ag}$ we can give an upper estimate for the maximum norm of the matrix $\mathbf{A}^{-1}$.

---

[3] Here we have used the well-known result for the spectral radius of an arbitrary matrix $\mathbf{C}$: in any induced norm the upper estimate $\rho(\mathbf{C}) < \|\mathbf{C}\|$ holds, which can be proven simply. Let $\mathbf{x} \neq \mathbf{0}$ is an arbitrary eigenvector of the matrix $\mathbf{C}$ with the eigenvalue λ. Then $|\lambda| \cdot \|\mathbf{x}\| = \|\lambda\mathbf{x}\| = \|\mathbf{Cx}\| \leq \|\mathbf{C}\| \cdot \|\mathbf{x}\|$, which implies the statement.

[4] The corresponding theorem states the following.

Theorem 7.1.5. *For an arbitrary matrix $\boldsymbol{C} \in \mathbb{C}^{n\times n}$ the relation $\boldsymbol{C}^k \to \boldsymbol{0}$ hold elementwise if and only if its spectral radius is less than 1, i.e., $\varrho(\boldsymbol{C}) < 1$. Moreover, the series*

$$\sum_{k=0}^{\infty} \boldsymbol{C}^k$$

is exactly convergent under this condition, and the sum of the series is the matrix $(\boldsymbol{E} - \boldsymbol{C})^{-1}$.

Theorem 7.1.7. *Let $\boldsymbol{A}$ be an M-matrix, and $\boldsymbol{g} > \boldsymbol{0}$ a vector with the property $\boldsymbol{Ag} > \boldsymbol{0}$. Then the estimate*

$$\|\boldsymbol{A}^{-1}\|_\infty \leq \frac{\|\boldsymbol{g}\|_\infty}{\min_i(\boldsymbol{Ag})_i} \tag{7.13}$$

holds.

Proof. Using the non-negativity of the matrix $\mathbf{A}^{-1}$, the proof follows from the relations

$$\|\mathbf{A}^{-1}\|_\infty \min_{i=1,\ldots,n}(\mathbf{Ag})_i = \|\mathbf{A}^{-1}\mathbf{e} \min_{i=1,\ldots,n}(\mathbf{Ag})_i\|_\infty \leq \|\mathbf{A}^{-1}\mathbf{Ag}\|_\infty = \|\mathbf{g}\|_\infty. \qquad \square$$

Example 7.1.8. *We consider the matrix in Example 7.1.2*

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

The vector $\boldsymbol{g} = [2,3,2]^T > \boldsymbol{0}$ is a suitable choice, since $\boldsymbol{Ag} = [1,2,1]^T > \boldsymbol{0}$. Hence we have the estimate

$$\|\boldsymbol{A}^{-1}\|_\infty \leq \frac{\|\boldsymbol{g}\|_\infty}{\min_i(\boldsymbol{Ag})_i} = \frac{3}{1} = 3,$$

which is really true, because $\|\boldsymbol{A}^{-1}\|_\infty = 2$.

Some properties of the matrix $\mathbf{v}_h$

We return to the consideration of the system (7.12). First we prove an important property of the matrix $\mathbf{v}_h$ in the system.

Theorem 7.1.9. *The matrix $\boldsymbol{v}_h$ defined by the formula* (7.12) *is an M-matrix.*

Proof. As we saw in the previous section, we have to show that in $\mathbb{R}^{N+1}$ there exists a positive vector $\mathbf{g}_h > 0$ such that the inequality $\mathbf{v}_h\mathbf{g}_h > 0$ holds. Let $\mathbf{g}_h$ be a vector whose i-th coordinate ($i = 0, 1, \ldots, N$) [5] is defined by

$$g_{h,i} = 1 + ih(l - ih). \tag{7.14}$$

[5]For simplicity, the first index is denoted by zero.

Then $\mathbf{g}_{h,i} \geq 1$ and $(\mathbf{v}_h\mathbf{g}_h)_0 = (\mathbf{v}_h\mathbf{g}_h)_N = 1$. We can easily verify the validity of the equality

$$-g_{h,i-1} + 2g_{h,i} - g_{h,i+1} = 2h^2$$

for any $i = 1, 2, \ldots, N-1$. Hence

$$(\mathbf{v}_h\mathbf{g}_h)_i = 2 + c(1 + ih(l - ih)), \quad i = 1, 2, \ldots, N-1.$$

This means that $(\mathbf{v}_h\mathbf{g}_h)_i \geq 1$ for any $i = 0, 1, 2, \ldots, N-1, N$. These relations imply that, using the previous notation $\mathbf{e} = [1, 1, \ldots, 1]^\top \in \mathbb{R}^{N+1}$, we have

$$\mathbf{g}_h \geq \mathbf{e} > 0, \text{ and } \mathbf{v}_h\mathbf{g}_h \geq \mathbf{e} > 0, \tag{7.15}$$

which proves our statement. □

Corollary 7.1.10. The matrix $\mathbf{v}_h$ is regular for any $h > 0$, and $\mathbf{v}_h^{-1} \geq 0$. Moreover, using the formula (7.13) of Theorem 7.1.7 we can give an upper estimate in the maximum norm for its inverse as follows.

$$\|\mathbf{v}_h^{-1}\|_\infty \leq \frac{\|\mathbf{g}_h\|_\infty}{\min_i(\mathbf{v}_h\mathbf{g}_h)_i} = \frac{\max_i g_{h,i}}{1}. \tag{7.16}$$

Using the relation for the arithmetic-geometric mean, we have

$$ih(l - ih) \leq \left(\frac{ih + (l - ih)}{2}\right)^2 = \frac{l^2}{4},$$

therefore $g_{h,i} \leq 1 + l^2/4$. Hence, using (7.16), the estimation

$$\|\mathbf{v}_h^{-1}\|_\infty \leq K = \frac{l^2 + 4}{4} \tag{7.17}$$

holds.

7.1.3 Convergence of the finite difference method

Let $P_h : C([0,1]) \to \mathbb{F}(\overline{\omega}_h)$ be a so-called *projection operator*, which is defined as $(P_h u)(x_i) = u(x_i)$ for any mesh-point $x_i \in \omega_h$. We denote by $e_h \in \mathbb{F}(\overline{\omega}_h)$ the function defined as $e_h = y_h - P_h u$, which is called *error function.*
Then

$$e_h(x_i) = y_h(x_i) - u(x_i) = y_i - u(x_i). \tag{7.18}$$

Definition 7.1.11. *The numerical method defined by the mesh-operator L_h is called* convergent in the maximum norm *when*

$$\lim_{h \to 0} \|e_h\|_\infty = 0. \tag{7.19}$$

If $\|e_h\|_\infty = \mathcal{O}(h^p)$ with some number $p \geq 1$, then we say that the convergence is of order p.

In the sequel we show that the numerical method (7.9) is convergent, and we define the order of the convergence, too.

Using the notation $e_h(x_i) = e_i$, from (7.18) we have $y_i = e_i + u(x_i)$. By substitution of this formula into the scheme (7.10), we get

$$\begin{gathered} -\frac{e_{i+1} - 2e_i + e_{i-1}}{h^2} + ce_i = \Psi_i^h, \quad i = 1, 2, \ldots, N-1, \\ e_0 = 0, \quad e_N = 0, \end{gathered} \tag{7.20}$$

where

$$\Psi_i^h = f_i + \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} - cu(x_i). \tag{7.21}$$

Since $f_i = f(x_i) = -u''(x_i) + cu(x_i)$, therefore

$$\Psi_i^h = \left(\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} - u''(x_i) \right) + \underbrace{(cu(x_i) - cu(x_i))}_{=0}. \tag{7.22}$$

Therefore,

$$\Psi_i^h = \mathcal{O}(h^2). \tag{7.23}$$

(In a more detailed form, by writing the leading constant, we have $\Psi_i^h = h^2 M_4/12 + \mathcal{O}(h^4)$, where the notation $M_4 = \max_{[0,l]} |u^{(4)}|$ is used.) We denote by $\Psi^h \in \mathbb{F}(\overline{\omega}_h)$ the mesh-function for which $\Psi^h(x_0) = \Psi^h(x_N) = 0$, and at mesh-points of ω_h it takes the values defined in (7.21). Then $\|\Psi^h\|_\infty = \mathcal{O}(h^2)$. We notice than the error equation (7.20) can be written in the form

$$\mathbf{v}_h \mathbf{e}_h = \Psi^h, \tag{7.24}$$

where $\mathbf{v}_h$ is the matrix of the form (7.12), $\mathbf{e}_h$ denotes the vector from $\mathbb{R}^{N+1}$ which corresponds to the error function e_h. Since $\mathbf{v}_h$ is regular, we may write $\mathbf{e}_h = \mathbf{v}_h^{-1} \Psi^h$. Hence, using the inequalities (7.17) and (7.23), we get

$$\|\mathbf{e}_h\|_\infty \leq \|\mathbf{v}_h^{-1}\|_\infty \|\Psi^h\|_\infty \leq K \cdot \mathcal{O}(h^2) = \mathcal{O}(h^2). \tag{7.25}$$

So $\lim_{h \to 0} \|\mathbf{e}_h\|_\infty = 0$. This proves the following statement.

Theorem 7.1.12. *Assume that the solution of problem* (7.1) $u(x)$ *is four times continuously differentiable. Then the numerical solution defined by* (7.5)-(7.6) *(or, alternatively, by the operator equation* (7.9)*) converges in second order in the maximum norm to the solution* $u(x)$*.*

Remark 7.1.1. In Theorem 7.1.12 the leading constant of the error function (i.e., the coefficient at h^2) is $M_4(l^2 + 4)/48$.

Based on the relation (7.22), Ψ_i^h shows in what order the values of the exact solution $u(x)$ satisfy the numerical scheme at the mesh-points x_i. Clearly, it is a basic demand that with h decreasing to zero (that is, when – in some sense – the mesh ω_h "tends" to he interval $[0, l]$), then Ψ_i^h should also tend to zero. Hence, the following definition is quite natural.

Definition 7.1.13. *We say that a numerical method with linear operator $\boldsymbol{v}_h$ is* consistent *when $\lim_{h\to 0} \Psi_i^h = 0$. When the method is consistent and $\Psi_i^h = \mathcal{O}(h^q)$ $(q \geq 1)$, then the method is called consistent with order q.*

According to our results, the method (7.9) is consistent with second order.

As we have seen in the proof of Theorem 7.1.12, the consistency in itself is not enough to prove the convergence: we have required some other property, namely, the estimate (7.17), which will be defined in an abstract way, as follows.

Definition 7.1.14. *We say that a numerical method with linear operator $\boldsymbol{v}_h$ is* stable*, when the operators (matrices) $\boldsymbol{v}_h$ are invertible, and there exists some constant $K > 0$, independent of h, such that the estimate*

$$\|\boldsymbol{v}_h^{-1}\|_\infty \leq K \tag{7.26}$$

is uniformly satisfied for any considered h.

Remark 7.1.2. For some numerical method the above stability property is valid only for sufficiently small values of h. This means that the there exists some number $h_0 > 0$ such that the estimate (7.26) holds for any $h < h_0$, but not for any $h > 0$. For the first case (i.e., when (7.26) holds for any $h < h_0$, only) the scheme is called *conditionally stable*. For the second case (i.e., when (7.26) holds for any $h > 0$) the scheme is called *unconditionally stable*. Since convergence examines the property of the scheme for small values of h, therefore the weaker property (the conditional stability) is enough (at least, theoretically) for our purposes. However, when h_0 is extremely small, (which may happen, e.g., in stiff problems), this may cause very serious problems. (E.g., when h_0 is close to the computer zero.)

The following statement establishes the connection between the notions of consistency, stability and convergence.

Theorem 7.1.15. *When a numerical method is consistent and stable, then it is convergent, and the order of convergence coincides with the order of consistency.*

The above theorem plays a central role in numerical analysis, and it is applied in many aspects. Therefore sometimes it is referred to as *the basic theorem of numerical analysis*.

We enclose an interactive animation, called *fdm_eng.exe*.The program can be downloaded from
`http://www.cs.elte.hu/~faragois/nummod_jegyzet_prog/`.

The image of this program on the screen can be seen in Figure 7.1.

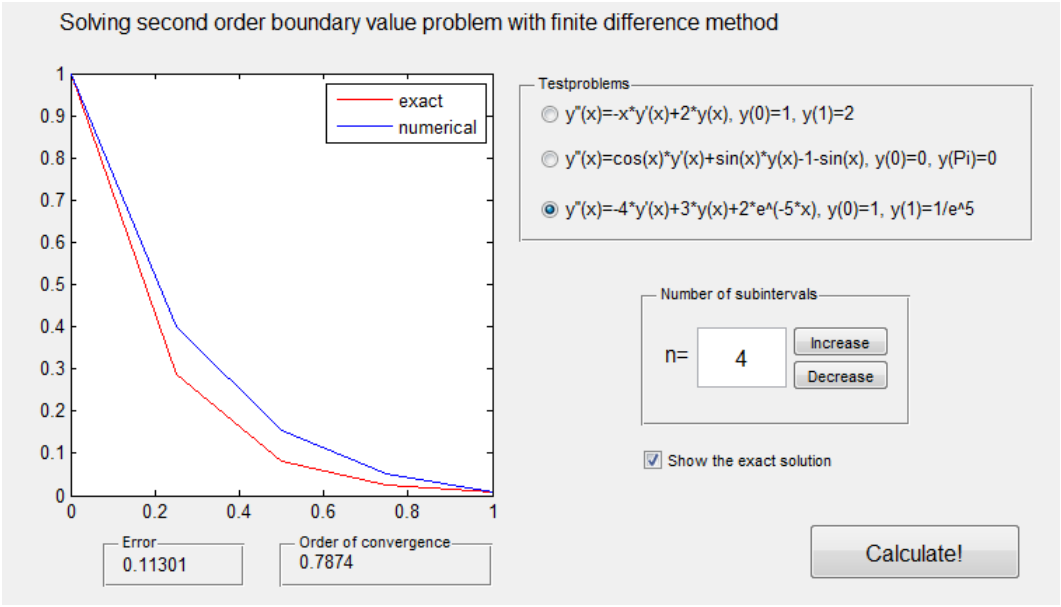

Figure 7.1: The image on the screen of the interactive program for the numerical solution of the boundary-value problem by using finite difference method.

Alternatively, this program suggests three initial-value problems as test problems. The chosen numerical method is the finite difference method. We can select the discretization step size by specifying the parameter n, which is the number of partitions of the interval. Pushing "Calculate" we get the result. The result is given graphically, and the error (in the maximum norm) is also indicated. By increasing n the order of the convergence is also shown.

7.1.4 Abstract formulation of the results

In the following we summarize the results of this section in an operator form. This allows us to apply our results to much more general settings, too.

Let $L : C^2[0,l] \to C(0,l)$ denote the operator:

$$(Lv)(x) = \begin{cases} -\dfrac{d^2v}{dx^2}(x) + cv(x), & \text{when } x \in (0,l); \\ v(x), & \text{when } x \in \{0;l\}, \end{cases} \tag{7.27}$$

moreover, $\tilde{f} \in C(0,l)$ the function

$$\tilde{f}(x) = \begin{cases} f(x), & \text{ha } x \in (0,l); \\ \mu_1, & \text{when } x = 0, \\ \mu_2, & \text{when } x = l. \end{cases} \tag{7.28}$$

Then our task is the solution of the operator equation[6]

$$Lu = \tilde{f}. \tag{7.29}$$

To this aim, on each mesh $(\omega_h) \subset [0,l]$ we define a problem of the form

$$L_h y_h = b_h, \tag{7.30}$$

where $b_h \in \mathbb{F}(\overline{\omega}_h)$ is given, $y_h \in \mathbb{F}(\overline{\omega}_h)$ is the unknown mesh-function, and $L_h : \mathbb{F}(\overline{\omega}_h) \to \mathbb{F}(\overline{\omega}_h)$ is a linear mesh-operator (matrix) of the form (7.7). Then, with the help of the projection operator $P_h : C[0,l] \to \mathbb{F}(\overline{\omega}_h)$ we have shown the following.

- The discrete problem is second order consistent, which means that the set of the operators $(L_h)_{h>0}$ approximates the operator L with the order h^2 on the exact solution u of the problem (7.29). This yield the validity of the estimate

  $$(L_h(P_h u))(x_i) - (Lu)(x_i) = \mathcal{O}(h^2) \tag{7.31}$$

  at any points $x_i \in \overline{\omega}_h$.

- The discrete problem is stable, that is, the operators L_h are regular and there exists a constant $K \geq 0$ such that the estimate (7.26) is valid.

We have shown that under these properties the convergence holds, i.e., the error function defined as $e_h = P_h u - y_h \in \mathbb{F}(\overline{\omega}_h)$ tends to zero in the maximum norm. Moreover, we have also shown that the convergence is of second order, i.e., $\|e_h\|_\infty = \mathcal{O}(h^2)$.

[6]We note that L is an operator of the type $L : C^2[0,l] \to C(0,l) \cap C(\{0,l\})$ and f is a mapping $\tilde{f} \in C(0,l) \cap C(\{0,l\})$. Since a function is always continuous at an isolated point, therefore the functions Lv and $\tilde{f}$ are automatically continuous at the points $x = 0$ and $x = l$.

7.2 Finite difference method

In this part we present the process of constructing a finite difference scheme for boundary value problems of the general form (5.7).

In the sequel we will use the well-known relations in order to approximate the derivatives in the differential equations. Namely, for any sufficiently smooth function w the following formulas are valid:

$$w'(x) = \frac{w(w+h) - w(x)}{h} - \frac{1}{2}hw''(\zeta_1) \tag{7.32}$$

$$w'(x) = \frac{w(x) - w(x-h)}{h} + \frac{1}{2}hw''(\zeta_2) \tag{7.33}$$

$$w'(x) = \frac{w(x+h) - w(x-h)}{2h} - \frac{1}{6}h^2w'''(\zeta_3) \tag{7.34}$$

$$w''(x) = \frac{w(x+h) - 2w(x) + w(x-h)}{h^2} - \frac{1}{12}h^2w^{(4)}(\zeta_4), \tag{7.35}$$

where $\zeta_i \in (x, x+h)$ $(i = 1, 2, 3, 4)$ are some fixed values.

Remark 7.2.1. The formulas (7.32)–(7.35) can be checked directly by the appropriate Taylor polynomials for the expressions $w(x+h)$ and $w(x-h)$ around the point x, and by using the Lagrange remainder. E.g.,

$$w(x+h) = w(x) + hw'(x) + \frac{1}{2}h^2w''(x) + \frac{1}{6}h^3w'''(x) + \frac{1}{24}h^4w^{(4)}(\zeta_4)$$

and

$$w(x-h) = w(x) - hw'(x) + \frac{1}{2}h^2w''(x) - \frac{1}{6}h^3w'''(x) + \frac{1}{24}h^4w^{(4)}(\zeta_4),$$

from which the statement follows directly.

This gives the idea of how to replace the derivatives of some (sufficiently smooth) function $w(x)$ at some fixed point $x^\star$ by some neighboring values of the function:

$$\begin{aligned}
w'(x^\star) &\simeq \frac{w(x^\star+h) - w(x^\star)}{h}, \\
w'(x^\star) &\simeq \frac{w(x^\star) - w(x^\star-h)}{h}, \\
w'(x^\star) &\simeq \frac{w(x^\star+h) - w(x^\star-h)}{2h}, \\
w''(x^\star) &\simeq \frac{w(x^\star+h) - w(x^\star) + w(x^\star-h)}{h^2}.
\end{aligned} \tag{7.36}$$

7.2.1 Boundary value problems in general form

Let us define a mesh on the interval $[a, b]$. For simplicity, we consider the uniform mesh

$$\overline{\omega}_h = \{x_i = a + ih, \quad i = 0, 1 \ldots, N + 1, \ h = (b - a)/(N + 1)\}, \tag{7.37}$$

where h is the *step size of the mesh*. We denote by ω_h the *inner points of the mesh*, i.e.,

$$\omega_h = \{x_i = a + ih, \quad i = 1, 2, \ldots, N, \ h = (b - a)/(N + 1)\}, \tag{7.38}$$

moreover, γ_h denotes *the boundary points of the mesh*, i.e.,

$$\gamma_h = \{x_0 = a, \ x_{N+1} = b\}. \tag{7.39}$$

We assume that the problem (5.7) has a unique solution $u(x)$ on $[a, b]$, that is, $u(x) \in C^2[a, b]$ is such a function[7] for which the relations

$$\begin{aligned} u''(x) =& f(x, u(x), u'(x)), \quad x \in (a, b), \\ & u(a) = \alpha, \quad u(b) = \beta \end{aligned} \tag{7.40}$$

hold. At the mesh-points x_i of the mesh $\overline{\omega}_h$ (7.40) yields the equations

$$\begin{aligned} u''(x_i) =& f(x_i, u(x_i), u'(x_i)), \quad i = 1, 2, \ldots, N \\ & u(t_0) = \alpha, \quad u(t_{N+1}) = \beta. \end{aligned} \tag{7.41}$$

Applying the approximations for the derivatives according to (7.36) in (7.41), for the unknown values y_i (which, according to our expectations, are some approximations to $u(x_i)$), we get the system of algebraic equations

$$\begin{aligned} \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} =& f(x_i, y_i, \frac{y_{i+1} - y_i}{h}), \quad i = 1, 2, \ldots, N \\ & y_0 = \alpha, \quad y_{N+1} = \beta. \end{aligned} \tag{7.42}$$

Solving the above system of algebraic equations for the $N+2$ unknown values $y_0, y_1, \ldots, y_{N+1}$, we obtain the approximations defined by the finite difference method. However, we do not know the answers to the following questions.

- Does the system (7.42) have a unique solution?

[7] As we can see from the relation (7.40), the slightly less strict condition $u(x) \in C^2(a, b) \cap C[a, b]$ would also be enough, but in real problems to guarantee the condition $u(x) \in C^2[a, b]$ is easier, and, if fact, it does not mean any restriction.

- If so, how can it be solved efficiently?
- Are the computed values y_i really close to the values $u(x_i)$? Can we estimate their distance?
- Will the sequence of numerical solutions converge on the interval $[a, b]$ to the exact solution when the mesh is refined (i.e., when h tends to zero)? If so, in what sense should the convergence be understood?

Remark 7.2.2. In the discretization (7.42) we replace the first derivative with the first formula in (7.36). When we use the second formula, we get the system

$$\begin{aligned} \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} =& f(t_i, y_i, \frac{y_i - y_{i-1}}{h}), \quad i = 1, 2, \ldots, N \\ y_0 = \alpha, \quad & y_{N+1} = \beta, \end{aligned} \tag{7.43}$$

and, when with the third one, then we get

$$\begin{aligned} \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} =& f(t_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}), \quad i = 1, 2, \ldots, N \\ y_0 = \alpha, \quad & y_{N+1} = \beta. \end{aligned} \tag{7.44}$$

We introduce the following notations:

$$y_{x,i} = \frac{y_{i+1} - y_i}{h}; \quad y_{\bar{x},i} = \frac{y_i - y_{i-1}}{h}; \quad y_{\mathring{x},i} = \frac{y_{i+1} - y_{i-1}}{2h}, \tag{7.45}$$

which are called right, left and central finite difference. Using the formulas (7.32) and (7.33), it is clear that the right and left side finite differences approximate in first order, while based on (7.34), the central finite difference approximates the first derivative at the point $x = x_i$ in second order.

Obviously,

$$y_{\bar{x}x,i} = \frac{y_{x,i} - y_{\bar{x},i}}{h} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \tag{7.46}$$

hence, due to the relation (7.35), $y_{\bar{x}x,i}$ approximates r the derivative $u''(x)$ at the point $x = x_i$ in second order. Using these notations, the difference schemes can be written in the following form :

- The scheme (7.42):

$$\begin{aligned} y_{\bar{x}x,i} =& f(t_i, y_i, y_{x,i}), \quad i = 1, 2, \ldots, N \\ y_0 = \alpha, \quad & y_{N+1} = \beta. \end{aligned} \tag{7.47}$$

- The scheme (7.43):

$$\begin{aligned} y_{\bar{x}x,i} =& f(t_i, y_i, y_{\bar{x},i}), \quad i = 1, 2, \ldots, N \\ & y_0 = \alpha, \quad y_{N+1} = \beta. \end{aligned} \tag{7.48}$$

- The scheme (7.44):

$$\begin{aligned} y_{\bar{x}x,i} =& f(t_i, y_i, y_{\hat{x},i}), \quad i = 1, 2, \ldots, N \\ & y_0 = \alpha, \quad y_{N+1} = \beta. \end{aligned} \tag{7.49}$$

7.2.2 Finite difference method for linear problems

In this part we consider the approximations by finite difference method of the linear boundary value problems having the form

$$\begin{aligned} u'' =& p(x)u' + q(x)u + r(x), \quad x \in [a, b], \\ & u(a) = \alpha, \quad u(b) = \beta \end{aligned} \tag{7.50}$$

and we give the answers to the questions raised in the previous section. As we have seen (c.f. Corollary 5.0.4), for the existence of a unique solution we assumed that $p, q, r \in C[a, b]$ and $q(x) > 0$ on the interval $[a, b]$, i.e., $\min_{[a,b]} q := q_{\min} > 0$. In the sequel we use the notations $p_i = p(t_i), q_i = q(t_i), r_i = r(t_i)$.

Theorem 7.2.1. *The discretization of the linear boundary value problem* (7.50) *by means of a finite difference method in the form* (7.47)-(7.49) *results in the system of linear algebraic equations*

$$\begin{aligned} a_i y_{i-1} + d_i y_i + c_i y_{i+1} =& -r_i, \quad i = 1, 2, \ldots, N \\ & y_0 = \alpha, \quad y_{N+1} = \beta, \end{aligned} \tag{7.51}$$

with some suitable chosen coefficients a_i, d_i and c_i, where, for sufficiently small h, for each discretization the equality

$$|d_i| - |a_i| - |c_i| = q_i \tag{7.52}$$

holds.

Proof. For problem (7.50) the discretization (7.47) yields the linear algebraic problem of the form

$$\begin{aligned} \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} =& p_i \frac{y_{i+1} - y_i}{h} + q_i y_i + r_i, \quad i = 1, 2, \ldots, N, \\ & y_0 = \alpha, \quad y_{N+1} = \beta. \end{aligned} \tag{7.53}$$

Hence, by the choice

$$a_i = -\frac{1}{h^2},\ d_i = \frac{2}{h^2} + q_i - \frac{1}{h}p_i,\ c_i = -\frac{1}{h^2} + \frac{1}{h}p_i \tag{7.54}$$

we get the problem (7.51). Therefore, for sufficiently small values of h we have

$$|d_i| - |a_i| - |c_i| = \frac{1}{h^2}\left((2 + h^2 q_i - hp_i) - 1 - (1 - hp_i)\right) = q_i.$$

We can easily see that for the discretization (7.48) the corresponding choice of the coefficients are

$$a_i = -\frac{1}{h^2} - \frac{1}{h}p_i,\ d_i = \frac{2}{h^2} + q_i + \frac{1}{h}p_i,\ c_i = -\frac{1}{h^2}, \tag{7.55}$$

while for the discretization (7.49) they are

$$a_i = -\frac{1}{h^2} - \frac{1}{2h}p_i,\ d_i = \frac{2}{h^2} + q_i,\ c_i = -\frac{1}{h^2} + \frac{1}{2h}p_i. \tag{7.56}$$

Hence, for both discretizations the equality (7.52) holds. □

Corollary 7.2.2. Obviously, system (7.51) can be written in the form

$$\begin{aligned} d_1 y_1 + c_1 y_2 &= -r_1 - a_1\alpha, \\ a_i y_{i-1} + d_i y_i + c_i y_{i+1} &= -r_i, \quad i = 2, 3, \ldots, N-1, \\ a_N y_{N-1} + d_N y_N &= -r_N - c_N\beta. \end{aligned} \tag{7.57}$$

This means that, in fact, the discretized problem is a linear system with N unknown values, and the matrix of the coefficients is a tridiagonal, strictly diagonally dominant matrix.

Let us denote by e_i the difference between the exact solution and the numerical solution at the point $x = x_i$ (i.e., $e_i = u_i - y_i$), and $\mathbf{e}_h \in \mathbb{F}(\omega_h)$ that mesh-function on ω_h called *error function* which is defined by these values (i.e., $\mathbf{e}_h(t_i) = e_i$).

Definition 7.2.3. *We say that a numerical method is convergent when the generated error function tends to zero, i.e., the relation*

$$\lim_{h\to 0} \boldsymbol{e}_h = 0 \tag{7.58}$$

holds. When $\boldsymbol{e}_h = \mathcal{O}(h^p)$, then the numerical method is said to be of order p.

In the next statement we consider the convergence and its order for the above defined finite difference schemes.

Theorem 7.2.4. *The finite difference approximation* (7.51) *with the coefficients* (7.54), (7.55) *and* (7.56), *is convergent to the solution of the linear boundary value problem* (7.50), *and*

1. *for the choice* (7.56) *second order convergence,*

2. *for the choices* (7.54) *and* (7.55) *first order convergence*

are valid.

Proof. Since the proofs are almost the same for both cases, we will only prove the first statement.

Using the approximation properties (7.34) and (7.35), problem (7.50) at the point $x = x_i$ can be re-written as

$$\begin{aligned}\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} - \frac{1}{12}h^2 u^{(4)}(\zeta_4^{(i)}) = \\ p_i\left(\frac{u_{i+1} - u_{i-1}}{2h} - \frac{1}{6}h^2 u'''(\zeta_3^{(i)})\right) + q_i u_i + r_i.\end{aligned} \tag{7.59}$$

Re-arranging the equality (7.59), we get

$$\left(-\frac{1}{h^2} + \frac{p_i}{2h}\right)u_{i+1} + \left(\frac{2}{h^2} + q_i\right)u_i + \left(-\frac{1}{h^2} - \frac{p_i}{2h}\right)u_{i-1} = -r_i - h^2 g_i, \tag{7.60}$$

where

$$g_i = \frac{1}{12}u^{(4)}(\zeta_4^{(i)}) - \frac{p_i}{6}u'''(\zeta_3^{(i)}).$$

Therefore, with the notations (7.56) the equation (7.60) can be written as

$$a_i u_{i-1} + d_i u_i + c_i u_{i+1} = -r_i - h^2 g_i. \tag{7.61}$$

This yields that the numerical approximations satisfy (7.51) with the coefficients from (7.56). Subtracting (7.51) from the equality (7.61), we get that the coordinates of the error function satisfy the system of linear algebraic equations

$$\begin{gathered}a_i e_{i-1} + d_i e_i + c_i e_{i+1} = -h^2 g_i, \quad i = 1, 2, \ldots, N \\ e_0 = e_{N+1} = 0,\end{gathered} \tag{7.62}$$

which has the same form as system (7.61).

Re-arranging the i-th equation, we get

$$d_i e_i = -a_i e_{i-1} - c_i e_{i+1} - h^2 g_i, \tag{7.63}$$

which, by estimation in absolute value, gives the inequalities

$$|d_i||e_i| \leq |a_i||e_{i-1}| + |c_i||e_{i+1}| + h^2|g_i| \leq (|a_i| + |c_i|)\,\|\mathbf{e}_h\|_\infty + h^2\|\mathbf{g}\|_\infty, \quad (7.64)$$

where $\mathbf{g}$ denotes the vector with coordinates g_i. We denote by i_0 the index for which $\|\mathbf{e}_h\|_\infty = |e_{i_0}|$. (Clearly $i_0 \neq 0$ and $i_0 \neq N+1$, since $e_0 = e_{N+1} = 0$.) Since the inequality (7.64) holds for any $i = 1, 2, \ldots N$, therefore it is true for the index $i = i_0$ as well. Hence we get

$$|d_{i_0}|\|\mathbf{e}_h\|_\infty \leq (|a_{i_0}| + |c_{i_0}|)\,\|\mathbf{e}_h\|_\infty + h^2\|\mathbf{g}\|_\infty. \quad (7.65)$$

The inequality (7.65) implies that the inequality

$$(|d_{i_0}| - |a_{i_0}| - |c_{i_0}|)\,\|\mathbf{e}_h\|_\infty \leq h^2\|\mathbf{g}\|_\infty \quad (7.66)$$

holds. So, by use of the property (7.52), we obtain the estimation

$$q_{i_0}\|\mathbf{e}_h\|_\infty \leq h^2\|\mathbf{g}\|_\infty. \quad (7.67)$$

Since $\min_{[a,b]} q := q_{\min} > 0$, therefore

$$\|\mathbf{e}_h\|_\infty \leq h^2\frac{\|\mathbf{g}\|_\infty}{q_{\min}} \leq \tilde{C}h^2, \quad (7.68)$$

where

$$\tilde{C} = (\frac{M_4}{12} + \frac{p_{\max}M_3}{6})/q_{\min}, \quad M_j = \max_{[a,b]}|u^{(j)}|, \quad p_{\max} = \max_{[a,b]}|p|.$$

But this yield that in case $h \to 0$ the error function tends to zero in second order. □

The following exercise is left to the Reader.

Example 7.2.5. *Prove the second statement of the theorem by using the proof of the first statement.*

Theorem 7.2.4 investigates the case where $q(t) > 0$ on $[a, b]$ in the linear boundary value problem (7.50). Hence, it is not applicable to the problem of parabolic throwing (5.5), where we have $p = q = 0$.

We consider the boundary value problem of the form

$$\begin{aligned} u'' &= r(x), \quad x \in (0, l), \\ u(0) &= \alpha, \quad u(l) = \beta. \end{aligned} \quad (7.69)$$

(For simplicity we take the interval $(0, l)$ instead of $[a, b]$, which does not mean any restriction of the generality.) In this case the mesh $\overline{\omega}_h$ is defined as

$$\overline{\omega}_h = \{x_i = ih, \quad i = 0, 1, \ldots, N+1, \ h = l/(N+1)\}. \tag{7.70}$$

Then system (7.51) for any discretization has the same form (7.57) with

$$a_i = -\frac{1}{h^2}, \ d_i = \frac{2}{h^2}, \ c_i = -\frac{1}{h^2}. \tag{7.71}$$

Hence the discrete problem is a system of linear algebraic equations of the form

$$\mathbf{A}_h \mathbf{y}_h = \mathbf{b}_h, \tag{7.72}$$

where $\mathbf{A}_h \in \mathbb{R}^{N \times N}$ is the matrix

$$\frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & \ldots & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & \ldots & 0 & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & \ldots & 0 & 0 & -1 & 2 \end{pmatrix} \tag{7.73}$$

and the vector $\mathbf{b}_h \in \mathbb{R}^N$ is defined as

$$\mathbf{b}_h = \begin{pmatrix} -r(t_1) + \alpha/h^2 \\ -r(t_i), \ i = 2, 3, \ldots, N-1 \\ -r(t_N) + \beta/h^2 \end{pmatrix}. \tag{7.74}$$

First we show that the above discretization is correct.

Theorem 7.2.6. *For any $h > 0$ the problem (7.72) has a unique solution.*

Proof. We show that for an arbitrary $h > 0$ the matrix $\mathbf{A}_h$ is regular, i.e., the value $\lambda = 0$ is not an eigenvalue.

To this aim, we define all eigenvalues $\lambda^{(k)}$ $(k = 1, 2, \ldots, N)$ of the matrix $\mathbf{A}_h$. Since $\mathbf{A}_h$ is symmetric, therefore any λ_k is real. By the definition of the eigenvalues, for the eigenvalues we have the equality $\mathbf{A}_h \mathbf{v}_k = \lambda_k \mathbf{v}_k$ with some non-zero vector $\mathbf{v}_k \in \mathbb{R}^N$, which (by omitting the notation k for the upper index) means that using the notations $v_0 = v_{N+1} = 0$ the coordinates of the eigenvectors $\mathbf{v}$ satisfy the equations

$$\frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} = \lambda v_i, \quad i = 1, 2, \ldots, N. \tag{7.75}$$

Hence we get a problem for a system of linear algebraic equations, having the form

$$\begin{gathered} v_{i-1} - 2(1 - 0.5\lambda h^2) v_i + v_{i+1} = 0, \quad i = 1, 2, \ldots, N, \\ v_0 = 0, \quad v_{N+1} = 0. \end{gathered} \tag{7.76}$$

We seek the solution of the problem (7.76) in the form

$$v_i = \sin(pt_i) \quad i = 0, 1, \ldots, N+1, \tag{7.77}$$

where $p \in \mathbb{R}$ is some constant, which will be defined later.

Substituting these values in the equations (7.76) for any $i = 1, 2, \ldots, N$, we get

$$\sin(p(t_i - h)) - 2(1 - 0.5\lambda h^2)\sin(pt_i) + \sin(p(t_i + h)) = 0. \tag{7.78}$$

Then, using the well-known elementary trigonometric identity $\sin(p(t_i - h)) + \sin(p(t_i + h)) = 2\sin(pt_i)\cos(ph)$, the equation (7.78) results in the equalities

$$2\sin(pt_i)\cos(ph) - 2(1 - 0.5\lambda h^2)\sin(pt_i) = 0, \quad i = 1, 2, \ldots, N, \tag{7.79}$$

i.e., we get the conditions

$$\big(2\cos(ph) - 2(1 - 0.5\lambda h^2)\big)\sin(pt_i) = 0, \quad i = 1, 2, \ldots, N. \tag{7.80}$$

Since $\sin(pt_i) = v_i$ and the eigenvector $\mathbf{v}$ is not the zero vector, therefore at least for one index i we have $v_i \neq 0$. Hence, due to (7.80), we have

$$2\cos(ph) - 2(1 - 0.5\lambda h^2) = 0. \tag{7.81}$$

This implies the condition

$$2\cos(ph) - 2(1 - 0.5\lambda h^2) = 0, \tag{7.82}$$

from which for the eigenvalue we obtain the formula

$$\lambda = \frac{2}{h^2}(1 - \cos(ph)) = \frac{4}{h^2}\sin^2\frac{ph}{2}. \tag{7.83}$$

We select the parameter p in such way that the second condition in (7.76) is also satisfied. Due to the choice in the form (7.77), we have $v_0 \equiv \sin(p \cdot 0) = 0$ and $v_{N+1} \equiv \sin(p \cdot l) = 0$. Clearly, the first condition is true for any p, and from the second condition we get $pl = k\pi$, that is, $p = p_k = k\pi/l$. Substituting this value into the relation (7.83), we get the eigenvalues of the matrix $\mathbf{A}_h$, which can be given by the formula

$$\lambda_k = \frac{4}{h^2}\sin^2\frac{k\pi h}{2l}, \quad k = 1, 2, \ldots, N. \tag{7.84}$$

Based on the relation (7.77) we can also express the eigenvectors $\mathbf{v}_k$:

$$\mathbf{v}_k = \left(\sin\frac{k\pi i h}{l}\right)_{i=1}^{N}, \quad k = 1, 2, \ldots, N. \tag{7.85}$$

Since λ_k depends on h, we use the notation $\lambda_k := \lambda_k(h)$. We can see that $\lambda_k(h) > 0$ for any $k = 1, 2, \ldots, N$, moreover, the smallest eigenvalue belongs to the index $k = 1$. This means that for any fixed h (i.e., on any fixed mesh) we have

$$\min_{k=1,2,\ldots,N} \lambda_k(h) = \lambda_1(h) = \frac{4}{h^2} \sin^2 \frac{\pi h}{2l}. \tag{7.86}$$

Let us introduce the new variable

$$s = \frac{\pi h}{2l}. \tag{7.87}$$

Since $h \leq l/2$, therefore $s \in (0, \pi/4]$. Then for the smallest eigenvalue we have the estimation

$$\lambda_1(s) = \frac{\pi^2}{l^2} \left(\sin s/s\right)^2, \quad s \in (0, \pi/4]. \tag{7.88}$$

We know that the function $\sin x/x$ is monotonically decreasing on the interval $(0, \pi/4]$.[8] Hence $\min_{s\in(0,\pi/4]} \lambda_1(s) = \lambda_1(\pi/4)$, i.e.,

$$\inf_{h>0} \lambda_1(h) = \lambda_1(\frac{l}{2}) = \frac{16}{l^2} \sin^2 \frac{\pi}{4} = \frac{8}{l^2}. \tag{7.89}$$

Therefore, for any $h > 0$ we have

$$\lambda_1(h) \geq \frac{8}{l^2}. \tag{7.90}$$

Hence the smallest eigenvalue of the matrix $\mathbf{A}_h$ is bigger than $\delta := 8/l^2 > 0$ for any $h > 0$, which proves our statement. □

Corollary 7.2.7. The above Theorem 7.2.6 and its proof directly implies the following statements.

1. The matrix $\mathbf{A}_h$ is regular, and the norm-estimate $\|\mathbf{A}_h^{-1}\|_2 \leq 1/\delta = l^2/8$ holds.

2. The matrix $\mathbf{A}_h$ is symmetric and positive definite.

3. From the formula (7.85) we can observe that $\mathbf{v}_1 > 0$. Since the relation $\lambda_1(h) > 0$ also holds, therefore, based on the obvious relation $\mathbf{A}_h\mathbf{v}_1 = \lambda_1(h)\mathbf{v}_1$ we obtain that with the vector $\mathbf{v}_1 > 0$ the relation $\mathbf{A}_h\mathbf{v}_1 > 0$ is true, i.e., $\mathbf{A}_h$ is an M-matrix.

[8] This can be simply verified by computing the derivative of this function.

Using the results we can show directly the convergence of the numerical discretization (7.57), (7.71) in the norm

$$\|v_h\|_{2,h}^2 := h\sum_{i=1}^{N} v_i^2, \tag{7.91}$$

which is given for the mesh-functions v_h, defined on the uniform mesh $\overline{\omega}_h := \{x_1, x_2, \ldots, x_N\}$ with mesh-size h.[9] For this case the matrix norm is defined as

$$\|\mathbf{A}\|_{2,h} = \sup_{\mathbf{v}\in\mathbb{R}^K} \frac{\|\mathbf{A}\mathbf{v}\|_{2,h}}{\|\mathbf{v}\|_{2,h}} = \sup_{\mathbf{v}\in\mathbb{R}^K} \frac{h\|\mathbf{A}\mathbf{v}\|_2}{h\|\mathbf{v}\|_2} = \|\mathbf{A}\|_2,$$

which means that it can be defined by the usual spectral norm.

In the following we consider the problem

$$\begin{gathered} \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} = r_i, \quad i = 1, 2, \ldots, N \\ y_0 = \alpha, \;\; y_{N+1} = \beta, \end{gathered} \tag{7.92}$$

and our aim is to show that its numerical solution converges to the solution of the problem (7.69) in the $\|\cdot\|_{2,h}$ norm in case $h \to 0$. We denote again by $e_i = u(t_i) - y_i$ the error function. Hence $y_i = -e_i + u(t_i)$, and by its substitution into the scheme (7.92) we get the following equations:

$$\begin{gathered} \frac{-e_{i-1} + 2e_i - e_{i+1}}{h^2} = r_i - \frac{u(t_{i-1}) - 2u(t_i) + u(t_{i+1})}{h^2} \equiv \psi_i, \quad i = 1, 2, \ldots, N \\ e_0 = 0, \quad e_{N+1} = 0, \end{gathered} \tag{7.93}$$

The problem (7.93) can be written in the form of the system of linear algebraic equations

$$\mathbf{A}_h \mathbf{e}_h = \boldsymbol{\psi}_h, \tag{7.94}$$

where $\mathbf{A}_h \in \mathbb{R}^{N\times N}$ is the matrix from (7.73), and the vector $\boldsymbol{\psi}_h \in \mathbb{R}^N$ is defined as $(\boldsymbol{\psi}_h)_i = \psi_i$, $i = 1, 2, \ldots, N$. Hence, from the (7.94) error equation we get

$$\mathbf{e}_h = \mathbf{A}_h^{-1} \boldsymbol{\psi}_h, \tag{7.95}$$

i.e., then in the discrete norm we have

$$\|\mathbf{e}_h\|_{2,h} \le \|\mathbf{A}_h^{-1}\|_{2,h} \|\boldsymbol{\psi}_h\|_{2,h} = \|\mathbf{A}_h^{-1}\|_2 \|\boldsymbol{\psi}_h\|_{2,h}. \tag{7.96}$$

[9] We can easily see that in case of fixed $a = x_1$ and $b = x_N$ this norm is equivalent to the discrete analogue of the $L_2[a, b]$ norm.

Due to (7.35) we have $\psi_i = \mathcal{O}(h^2)$, therefore

$$\|\boldsymbol{\psi}_h\|_{2,h}^2 = h\sum_{i=1}^{N}\psi_i^2 = h\sum_{i=1}^{N}\mathcal{O}(h^4) = hN\mathcal{O}(h^4) = \mathcal{O}(h^4),$$

because $hN = const$. Therefore the estimation $\|\boldsymbol{\psi}_h\|_{2,h} = \mathcal{O}(h^2)$ is true. On the other side, based on Corollary 7.2.7, we have $\|\mathbf{A}_h^{-1}\|_2 \leq l^2/8$. Hence, using the equation (7.96), we have

$$\|\mathbf{e}_h\|_{2,h} = \mathcal{O}(h^2). \tag{7.97}$$

This proves the validity of the following statement.

Theorem 7.2.8. *In case $h \to 0$, the numerical solution defined by the scheme* (7.92) *converges in second order to the solution of the boundary value problem* (7.69) *in the $\|\cdot\|_{2,h}$-norm.*

Chapter 8

Solving boundary value problems with MATLAB

Typically, MATLAB recommends the routine so-called BVP4C for solving boundary value problems. This routine solves such problems under rather general conditions. We can solve the problems with non-separate boundary conditions (i.e., when the values of the unknown solution are known not only at the different end-points, but when we know some linear combinations of these values). We can also investigate the problems when the boundary conditions depend on some parameters, too. The applied algorithm is based on the collocation method, and, as a result, we obtain a system of non-linear algebraic equations. For solving this system, Newton's method is applied, which requires the knowledge of the partial derivatives. In the program these derivatives are defined approximately, by using the finite difference method. (The details of the program can be found under the link
`http://200.13.98.241/~martin/irq/tareas1/bvp_paper.pdf`
in the description, entitled "Solving Boundary Value Problems for Ordinary Differential Equations in MATLAB with BVP4C (Lawrence F. Shampine, Jacek Kierzenka, Mark W. Reichelt). We also recommend the link
`http://www.mathworks.com/matlabcentral/fileexchange/3819`
where a tutorial material is available for BVP4C, and one can found many useful applications, too. We suggest visiting the website under the link
`http://www.mathworks.com/help/techdoc/ref/bvp4c.html`
where besides the description of the program one can find different real applications, too.

For the two basic numerical methods (shooting method and finite difference method), which were investigated in our textbook, the Reader can create their own program. In the sequel we describe the built-in boundary value problem

solver BVP4C, and we also deal with the description and numerical solution of some model problems.

8.1 A built-in boundary value problem solver: bvp4c

The built-in MATLAB library BVP4C consists a finite difference method that implements the 3-stage Lobatto IIIa formula. This is a collocation formula and the collocation polynomial provides a C^1-continuous solution that is fourth-order accurate uniformly in the interval of integration. Mesh selection and error control are based on the residual of the continuous solution. The collocation technique uses a mesh of points to divide the interval of integration into subintervals. The solver determines a numerical solution by solving a global system of algebraic equations resulting from the boundary conditions, and the collocation conditions imposed on all the subintervals. The solver then estimates the error of the numerical solution on each subinterval. If the solution does not satisfy the tolerance criteria, the solver adapts the mesh and repeats the process. The user must provide the points of the initial mesh as well as an initial approximation of the solution at the mesh points.

The basic syntax of the BVP solver BVP4C is

$$sol = bvp4c(odefun, bcfun, solinit).$$

The input arguments are:

- ODEFUN: Function that evaluates the differential equations. It has the basic form $dydx = odefun(x, y)$ where x is a scalar, and $dydx$ and y are column vectors. ODEFUN can also accept a vector of unknown parameters and a variable number of known parameters.

- BCFUN: Function that evaluates the residual in the boundary conditions. It has the basic form
$$res = bcfun(ya, yb),$$
where ya and yb are column vectors representing $y(a)$ and $y(b)$, and res is a column vector of the residual in satisfying the boundary conditions. The function BCFUN can also accept a vector of unknown parameters and a variable number of known parameters.

- SOLINIT: This forms the initial guess for BVP4C. Structure with fields x and y:

1. x: Ordered nodes of the initial mesh. Boundary conditions are imposed at $a = solinit.x(1)$ and $b = solinit.x(end)$.
2. y: Initial guess for the solution with $solinit.y(:, i)$ a guess for the solution at the node $solinit.x(i)$.

   The structure can have any name, but the fields must be named x and y. It can also contain a vector that provides an initial guess for the unknown parameters. We can form SOLINIT with the helper function BVPINIT. (We note that $solinit = bvpinit(sol, [anewbnew], parameters)$ forms SOLINIT as described above, but uses parameters as a guess for unknown parameters in SOLINIT. See the BVPINIT reference page for details.)

The output argument sol is a structure created by the solver. In the basic case the structure has fields x, y , and yp.

- $sol.x$: Nodes of the mesh selected by BVP4C.
- $sol.y$: Approximation to $y(x)$ at the mesh points of $sol.x$.
- $sol.yp$: Approximation to $y'(x)$ at the mesh points of $sol.x$.
- $sol.parameters$: Value of unknown parameters, if present, found by the solver.

The function DEVAL uses the output structure sol to evaluate the numerical solution at any point from $[a, b]$.

For more advanced applications, we can also specify as input arguments solver options and additional known parameters.

- *options*: Structure of optional parameters that change the default integration properties. This is the fourth input argument

$$sol = bvp4c(odefun, bcfun, solinit, options).$$

- $p1, p2, \ldots$: Known parameters that the solver passes to ODEFUN and BCFUN

$$sol = bvp4c(odefun, bcfun, solinit, options, p1, p2, \ldots)$$

  The solver passes any input parameters that follow the options argument to ODEFUN and BCFUN every time it calls them. (We set options = [ ] as a placeholder if we set no options.) In the ODEFUN argument list, known

parameters follow x, y, and a vector of unknown parameters (parameters), if present.

$$dydx = odefun(x, y, p1, p2, \dots)$$

$$dydx = odefun(x, y, parameters, p1, p2, \dots)$$

In the BCFUN argument list, known parameters follow ya, yb, and a vector of unknown parameters, if present

$$res = bcfun(ya, yb, p1, p2, \dots)$$

$$res = bcfun(ya, yb, parameters, p1, p2, \dots)$$

.

Remark 8.1.1. We have prepared an animation *bvp.gif* for the finite difference method which can be seen the convergence of the numerical solution.

`../animations/bvp.gif`

8.2 Application: the mathematical description

In the following we apply this routine to a model problem, namely, to the numerical solution of stationary heat distribution in homogeneous rod.

We assume that a long, thin rod is located between two walls having stationary temperature. The thickness of the rod is negligible w.r.t. the length, therefore the radial temperature change (radial radiation) is also negligible. Hence the temperature depends on the one dimensional coordinate x, only. The heat flow in the rod occurs due to the longitudinal conduction, on the other hand, due to convection between the rod and the rod surrounding gas, which has constant temperature. Our task is to find the stationary heat distribution.

First we write the balance equation on the element of length Δx. We denote by $q(x)$ the heat flux at the point x. (Its unit is $J/(m^2 s)$].) A_c denotes the cross-sectional area $[m^2]$), i.e., $A_c = \pi r^2$, where r is the radius of the rod. Let D_{kon} denote the convection coefficient of the heat transfer ($J/(Km^2 s)$), where K is the temperature in Kelvin units, A_s is the surface (m^2), i.e., $A_s = 2\pi r \Delta x$, and T_∞ denotes the temperature of the surrounding gas (K). $T(x)$ denotes the temperature at the point x. Then, based on the balance equation, we get the relation

$$q(x)A_c = q(x + \Delta x)A_c - D_{kon}A_s(T_\infty - T(x)), \tag{8.1}$$

where the left side contains the ingoing heat quantity, and the right side the difference between the outgoing heat quantity and the loss of the heat flow in the radial direction, respectively. Dividing the equality (8.1) by the volume of an elementary unit ($\pi r^2 \Delta x$), after re-arranging the expression we obtain the equation

$$-\frac{q(x+\Delta x)-q(x)}{\Delta x}+\frac{2D_{kon}}{r}(T_\infty - T(x))=0. \tag{8.2}$$

Hence, passing to the limit $\Delta x \to 0$ in (8.2), we obtain

$$-q'(x)+\frac{2D_{kon}}{r}(T_\infty - T(x))=0. \tag{8.3}$$

According to the Fourier law, for the flux $q(x)$ the relation $q(x) = -D_{hcc}T'(x)$ holds, where D_{hcc} $[J/(Kms)]$ is the heat conduction coefficient. Then the equation (8.3) can be written as

$$T''(x)+D(T_\infty - T(x))=0, \tag{8.4}$$

where

$$D=\frac{2D_{kon}}{rD_{hcc}}$$

is the heat dissipation constant $[m^{-2}]$. Denoting by L the length of the rod, we have

$$T(0)=T_0, \quad T(L)=T_1, \tag{8.5}$$

where T_0 and T_1 denote the temperatures on the left and right side walls, respectively.

Hence, the mathematical model can be written as a two-point boundary value problem, having the form (8.4)–(8.5).

The exact analytical solution of problem (8.4)–(8.5) can be given. The equation can be easily transformed into the form

$$T''(x)-DT(x))=-DT_\infty. \tag{8.6}$$

The solution of this problem is the sum of the general solution of the homogeneous equation and of some particular solution of the non-homogeneous problem. Since the characteristic equation for the homogeneous equation $T''(x)-DT(x))=0$ is $\mu^2 - D = 0$, therefore, using the notation $\lambda = \pm\sqrt{D}$, the general solution of the homogeneous problem is $T_{hom}(x) = C_1 e^{\lambda x} + C_2 e^{-\lambda x}$, where C_1 and C_2 are arbitrary constants. Let us notice that the constant function $T_{part}(x) = T_\infty$ is a particular solution. Hence the general solution of the equation (8.6) is

$$T(x) = C_1 e^{\lambda x} + C_2 e^{-\lambda x} + T_\infty. \tag{8.7}$$

The arbitrary constants can be defined from the boundary conditions (8.5):

$$\begin{aligned} C_1 =& \frac{(T_0 - T_\infty)e^{-\lambda L} - (T_1 - T_\infty)}{e^{-\lambda L} - e^{\lambda L}}, \\ C_2 =& \frac{-(T_0 - T_\infty)e^{\lambda L} + (T_1 - T_\infty)}{e^{-\lambda L} - e^{\lambda L}}. \end{aligned} \tag{8.8}$$

In the sequel we define the parameters as follows: $L = 10$, $D_{kon} = 1$, $D_{hov} = 200$, $r = 0.2$, $T_\infty = 200$, $T_0 = 300$, $T_1 = 400$. (We have used the the same units as before.) Then $D = 0.05$, and the exact solution of the problem is the function [1]

$$T(x) = 20.4671e^{\sqrt{0.5}x} + 79.5329e^{-\sqrt{0.5}x} + 200. \tag{8.9}$$

8.3 Application: MATLAB program for the shooting method

Now we pass on to the numerical solution of the model problem by using MATLAB.

We apply the shooting method to the problem (8.4)–(8.5). In the first step we re-write the scalar ordinary differential equation of second order in the form of a system of first order ODE's with two unknown functions. This means that we consider the system

$$\begin{aligned} T'(x) &= z(x), \\ z'(x) &= -D(T_\infty - T(x)). \end{aligned} \tag{8.10}$$

For the initial values we set

$$T(0) = T_0, \quad z(0) = z_0, \tag{8.11}$$

where T_0 is known, and the value z_0 is chosen in such a way that for the solution $T(x)$ of the initial value problem (8.10)–(8.11) the equality $T(L) = T_1$ is satisfied.

According to the algorithm of the shooting method, the MATLAB code consists of two functions. The routine BVPS defines the numerical solution of the initial value problem with some given initial value z_0, using the explicit Euler method.

[1] In MATLAB we do not use the values for the constants in formulas (8.9), but we define them based on the formula (8.7).

In this part the input parameters are the number of partitions of the interval $[0, L]$ (Nx), and the initial value of the component z of the unknown vector-function is z_0. The output parameters are the temperature (TL) at the end-point ($x = L$), the temperature at the mesh-points ($Tvect$) and the coordinates of the mesh ($xvect$).

The routine is the following:

```
function [TL,Tvect,xvect] = bvpsee(Nx,z0)
%
% Shooting method for computing the stationary heat distribution
% of a rod of length L by numerical method
% We solve the following two-point boundary value problem:
%
% d^2T
% ----- +D(Tinf - T) = 0,    T(0) = T0,    T(L) = T1
% dx^2
% For solving the initial value problem the
% explicit Euler method is applied.
% Nx:  number of the space division
% z0:  the initial steepness
T0 = 300; % boundary value on the left side
T1 = 400; % boundary value on the right side
Tinf = 200; % outside temperature
D = 0.05;  % constant
L = 10; % length of the rod
xs = 0; %coordinate of the initial point
T = T0; % initial condition
deltax=L/Nx; Tvect(1) = T; xvect(1) = xs; z=z0;
for i=2:Nx+1
    dTdx = z;
    dzdx = -D * (Tinf - T);
    T = T + dTdx * deltax; % Euler method
    z = z + dzdx * deltax;
    xvect(i) = xs + (i-1)*deltax; Tvect(i) = T;
end;
TL=T;
end
```

Using the above routine we can compute the numerical solution of an initial value problem with some given initial condition. Obviously, this is not sufficient for our purposes, since the computed numerical approximation TL differs from the fixed value $T1$. Following the algorithm of the shooting method, by

using the BVPS routine we define the temperature at the end-point TL for different initial values $z(0)$, and, by using these values, we define the difference $\varepsilon(z(0)) = TL(z(0)) - T1$, obtained for the different values $z(0)$. We search for the value $z^\star(0)$ such that the relation $\varepsilon(z^\star(0)) = 0$ holds. Therefore we define an interpolation polynomial for the points $(z(0), \varepsilon(z(0)))$, and for $z^\star(0)$ we accept point, where this polynomial turns into zero. In the MATLAB code these steps are realized by using the inverse interpolation, given by (6.14).

The above steps are carried out in the routine SHOOTING, which has the following input parameters.

- Nx: number of partitions of the interval $[0, L]$,
- *zstart*: the first guess for the initial value $z(0)$ (which is the gradient of the unknown function T at the point $x = 0$),
- *deltaz*: the increment for the different values $z(0)$,
- Nz: from the initial value *zstart* we define further Nz values to the right and to the left directions, namely, we define the value TL by the initial values $z(0) = zstart \pm k \cdot deltaz$ $(k = 1, 2, \ldots, Nz)$, too. (This produces the basic interpolation points.)

The routine can optionally print the numerical solution, and, when the exact solution is known in certain test problem, we can print the error in the maximum norm, and we can also draw the exact and approximate solutions.

```
function shooting(Nx,zstart,deltaz,Nz)
%
% Shooting method for computing the stationary heat distribution
% of a rod of length L by numerical method
% We solve the following two-point boundary value problem:
%
% d^2T
% ----- +D(Tinf - T) = 0,   T(0) = T0,   T(L) = T1
% dx^2
% For the solution of the initial value problem
% the routine BVPS is applied.
% Nx:  number of the space division
% z0:  the initial steepness
T0 = 300; % boundary value on the left side
T1 = 400; % boundary value on the right side
Tinf = 200; % outside temperature
D = 0.05;  % constant
```

```
L = 10; % length of the rod
xs = 0; %coordinate of the initial point
T = T0; % initial condition
deltax=L/Nx; zv(Nz+1)=zstart; z=zv(Nz+1);
[T,Tvect,xvect]=bvpsee(Nx,z);Tvegpont(Nz+1)=T;
for i=1:Nz
    zv(i)=zstart-(Nz+1-i)*deltaz; z=zv(i);
    [T,Tvect,xvect]=bvpsee(Nx,z);Tvegpont(i)=T;
    zv(Nz+1+i)=zstart+i*deltaz; z=zv(Nz+1+i);
    [T,Tvect,xvect]=bvpsee(Nx,z);Tvegpont(Nz+1+i)=T;
end
for i=1:2*Nz+1
    Tvegpont(i); zv(i);
end
% For finding the root we apply the inverse interpolation.
Tint=Tvegpont-T1; z=interp1(Tint,zv,0);
fprintf('Steepness:  %fn',z)
[Tfinal,Tvect,xvect]=bvpsee(Nx,z);
% In the array 'Meg' we collect the numerical solution
% and, in case of necessity, we print out.
Meg=ones(Nx+1,3);
for i=1:Nx+1
    Meg(i,1)=i; Meg(i,2)=xvect(i); Meg(i,3)=Tvect(i);
end
disp('coord.  of the mesh-point.  temperature')
disp(Meg)
plot(xvect,Tvekfinal);
% If we know the exact solution, here we define
% in the function PRECISESHOOTING.
% Then the error can be calculated, and, optionally we can draw
% the exact and the numerical solutions.
[Tpontos,zpontos]=preciseshooting(Nx,xvect);
hiba=norm(Tvect-Tpontos,inf);
disp('step size and the error in maximum norm:'),deltax, error
i=1:Nx+1;
plot(i,Tvect,'r', i, Tpontos,'b') xlabel('rod'),
ylabel('shooting method (red),
exact solution (blue)')
```

Based on formula (8.9), we know the exact solution. This allows us to verify the shooting method with different parameters. The maximum norm

Δx	1	0.1	0.01	0.001	0.0001
e_h	$64.0278e+000$	$4.6133e-001$	$4.6786e-002$	$4.6852e-003$	$4.6859e-004$
order	$n.a.$	$1.1453e-001$	$1.0142e-001$	$1.0014e-001$	$1.0001e-001$

Table 8.1: The maximum norm of the error of the shooting method, combined with the explicit Euler method, applied to the test problem, and the order of the convergence.

Δx	1	0.1	0.01	0.001	0.0001
e_h	$6.3073e-001$	$6.6929e-003$	$6.7386e-005$	$6.7433e-007$	$6.7446e-009$
order	$n.a.$	$1.0611e-002$	$1.0068e-002$	$1.0007e-002$	$1.0002e-002$

Table 8.2: Maximum norm of the error for the shooting method combined with the improved Euler method and the order of convergence.

of the error vector for different Δx is given in Table 8.1. The exact and the numerical solutions on the different meshes are illustrated in Figures 8.1-8.4.

Remark 8.3.1. The test problem was solved with parameters $Nz = 4$ and $zstart = -12$. When choosing larger Nz the numerical result does not change significantly. The reason is that the test problem is linear, i.e., according to Section 6.3, based on the knowledge of two data, the initial slope can be calculated directly. Hence, there is no need for giving more points, and, in fact the choice $Nz = 2$ is sufficient. The Figures 8.1 and 8.5 well reflect this property. (These figures were obtained using the same space partition, but different values of Nz. (Namely, we set $Nz = 2$ and $Nz = 10$.)

Remark 8.3.2. When values $zstart$ are very far, then, in case of a few interpolation basic points only, it may happen that the inverse interpolation does not work. (The zero value lays out of the basic points.) In such a case we suggest to perform some independent runs with the routine BVPS, and define approximately the unknown value z.

Remark 8.3.3. Replacing the explicit Euler method by the improved Euler method (which has second order accuracy), we obtain the errors given in Table 8.2. As expected, the method is convergent in second order.

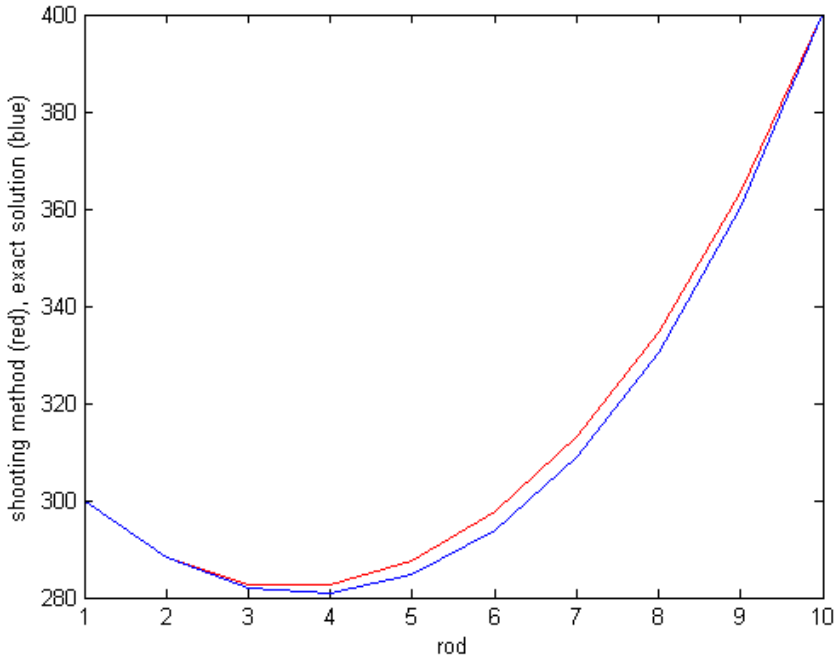

Figure 8.1: Stationary heat distribution in the rod of length $L = 10m$ using the uniform mesh with 10 mesh-points.

8.4 Application: MATLAB program for the finite difference method

In the following we deal with the numerical solution of problem (8.4)–(8.5) using the finite difference method. Not going into the details of the algorithm, we give the routine VDM1, which solves the above problem by using the finite difference method. In this program the input parameter is the number of the partition (`Nx`) for the rod of length `L`. The output parameters are the coordinates of the mesh-points (`xvect`) and the numerical solution at these points is stored in the vector `Tsol`.

```
function[xvect,Tsol]=vdm1(Nx)

Ta = 300; % boundary value on the left side
Tb = 400; % boundary value on the right side
Tinf = 200; % outside temperature
c = 0.05; % constant
```

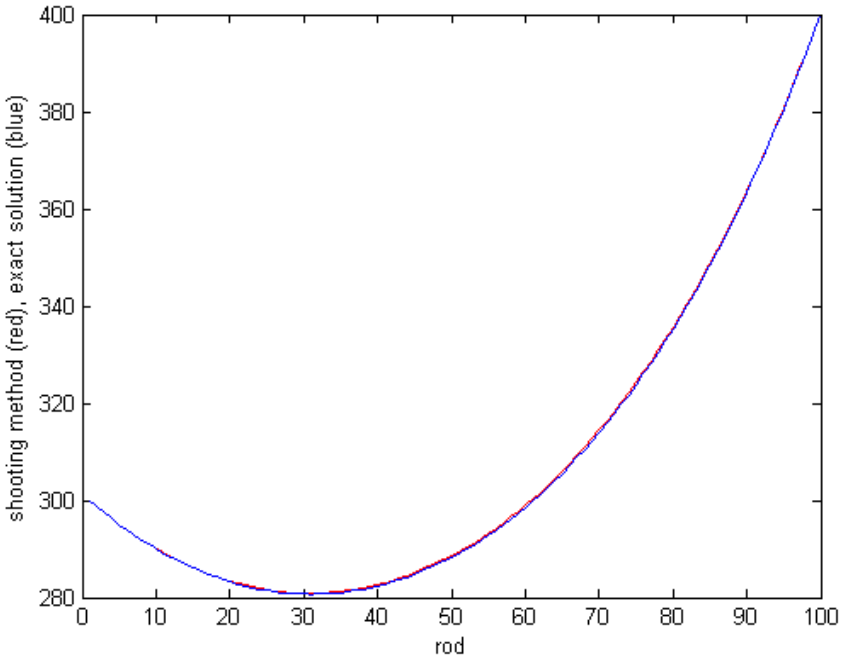

Figure 8.2: Stationary heat distribution in the rod of length $L = 10m$ using the uniform mesh with 100 mesh-points.

```
L = 10; % length of the rod
ndivs = Nx; nunknowns = ndivs - 1; deltax = L/ndivs;

A = -(2 + deltax^2*c); B = -deltax^2*c*Tinf;

for i=1:Nx+1, % generating the mesh-points
xvect(i)=(i-1)*deltax;
end;
```

Starting the compilation of the system of linear algebraic equations

```
matrix = zeros(nunknowns);

matrix(1,1) = A; % first row
matrix(1,2) = 1; rhs(1)= B - Ta;

for i = 2:nunknowns - 1 % intermediate rows
```

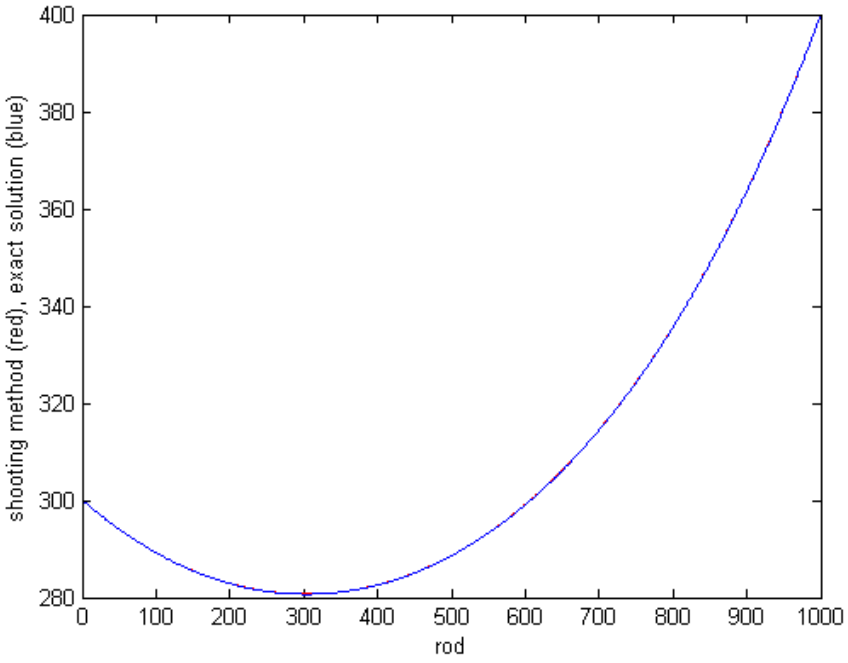

Figure 8.3: Stationary heat distribution in the rod of length $L = 10m$ using the uniform mesh with 1000 mesh-points.

```
matrix(i,i-1) = 1; matrix(i,i) = A;

matrix(i,i+1) = 1;

rhs(i)= B; end;

matrix(nunknowns, nunknowns-1) = 1; % last row
matrix(nunknowns, nunknowns) = A; rhs(nunknowns)= B - Tb;
T = matrix\rhs'; % solve for unknowns
Tsol(1)= Ta; % reconstruct full vector
Tsol(2:1 + nunknowns) = T(:); Tsol(nunknowns + 2) = Tb;
end
```

Our results for the test problem are summarized in Table 8.3. For the first three cases (i.e., for $Nx = 3, 6, 9$) the finite difference solution with the exact solution are shown in Figures 8.6-8.8.

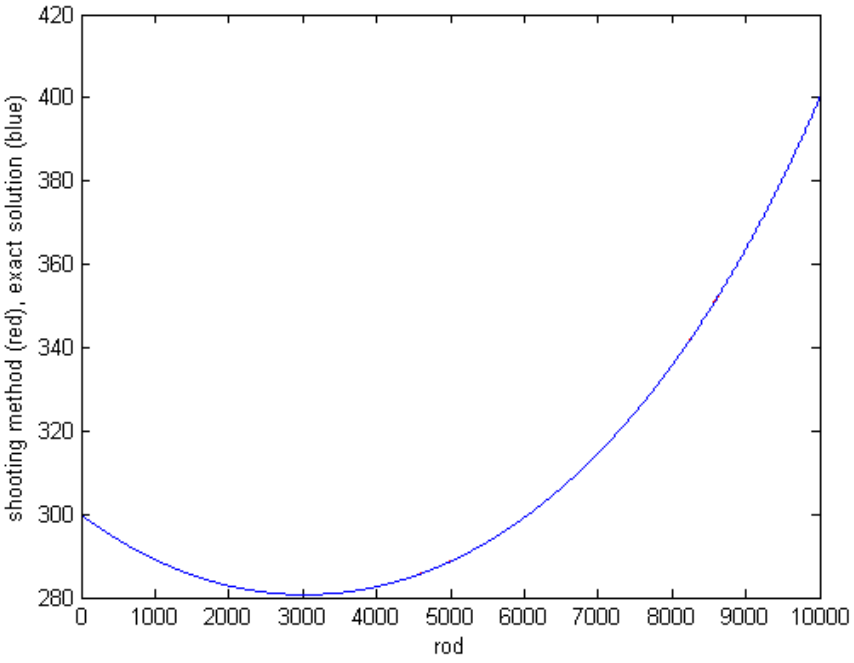

Figure 8.4: Stationary heat distribution in the rod of length $L = 10m$ using the uniform mesh with 10000 mesh-points.

Finally, we compare the discussed methods on the same mesh. Let $Nx = 8$, i.e., we use an equidistant mesh with 9 points and step-size $\Delta x = 1.25$. We compare the shooting method combined with the explicit Euler method and the improved Euler method, and the finite difference method. The results are contained in Table 8.4. The solutions are shown in Figure 8.9. Since the solutions are very close to each other, therefore we enlarged some part in Figure 8.10.

# partition	error in max. norm	order
3	$1.7002e+000$	
6	$4.5604e-001$	$2.6823e-001$
12	$1.1647e-001$	$2.5540e-001$
24	$2.9205e-002$	$2.5074e-001$
48	$7.3158e-003$	$2.5050e-001$
96	$1.8293e-003$	$2.5004e-001$
192	$4.5734e-004$	$2.5001e-001$
384	$1.1434e-004$	$2.5000e-001$
768	$2.8582e-005$	$2.4999e-001$
1536	$7.1573e-006$	$2.5041e-001$

Table 8.3: Errors of the finite difference method with halving step-size. The column shows the order. (The theoretical order is 0.25.)

mesh-points	explicit Euler	improved Euler	finite difference	exact solution
0	300.0000	300.0000	300.0000	300.0000
1.2500	287.2008	287.6551	287.3173	287.3173
2.5000	282.2141	282.0058	281.4562	281.4562
3.7500	284.0400	282.6293	281.9589	281.9589
5.0000	292.2889	289.5832	288.8646	288.8646
6.2500	307.1033	303.4096	302.7129	302.7129
7.5000	329.1279	325.1783	324.5856	324.5856
8.7500	359.5199	356.5687	356.1916	356.1916
10.0000	400.0000	400.0000	400.0000	400.0000

Table 8.4: The results of the different methods for the test problem for the partition $Nx = 8$. The exact solution is red, the shooting method with the explicit Euler method is green, the shooting method with the improved Euler method is black, the finite difference method is blue.

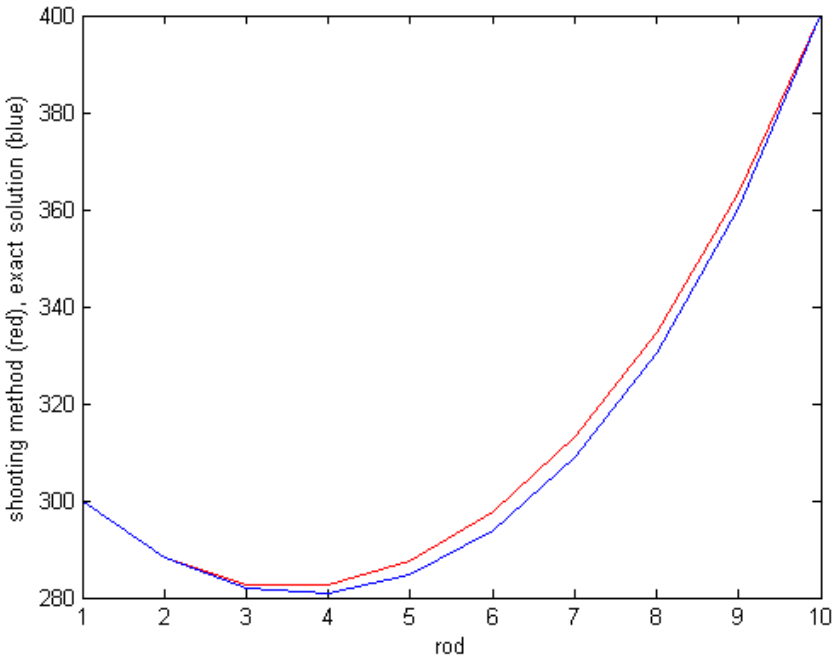

Figure 8.5: Stationary heat distribution in the rod of length $L = 10m$ using the uniform mesh with 10 mesh-points and using interpolation based on two points ($Nz = 2$).

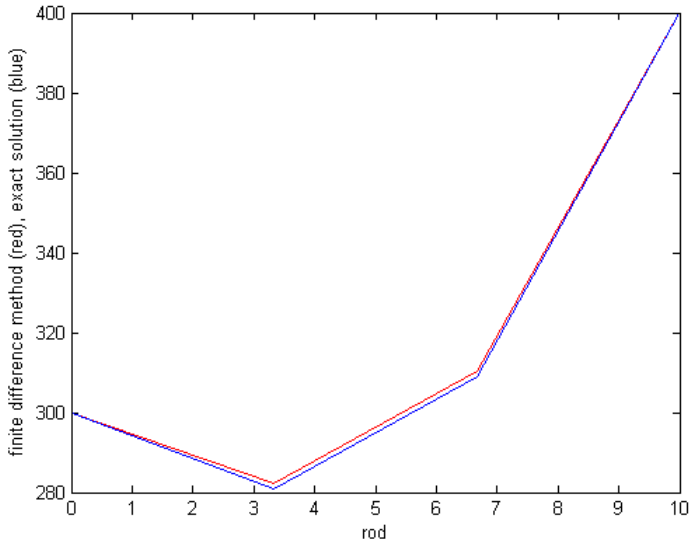

Figure 8.6: The heat distribution in a rod of length $L = 10m$ by finite difference method, using a uniform mesh with 4 points.

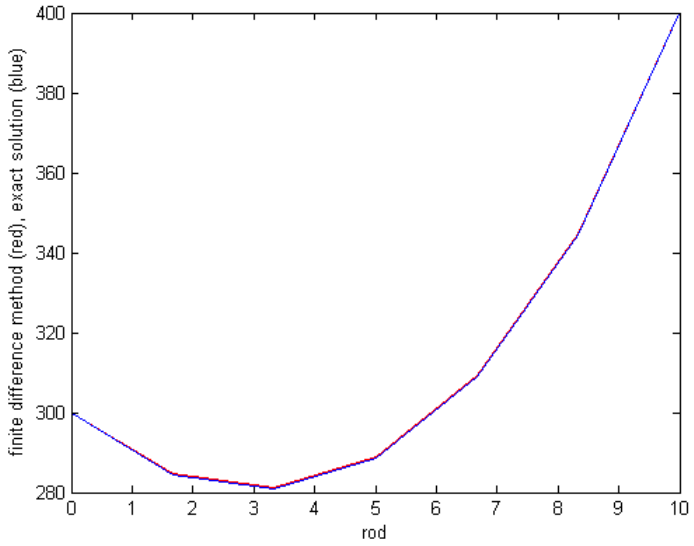

Figure 8.7: The heat distribution in a rod of length $L = 10m$ by finite difference method, using a uniform mesh with 7 points.

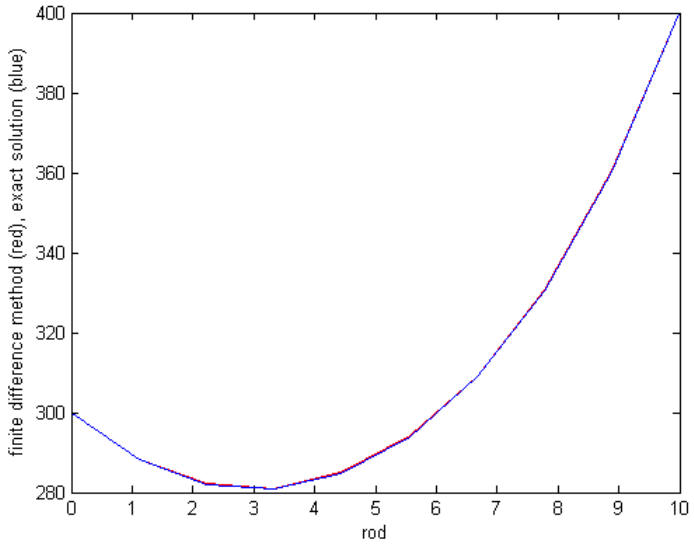

Figure 8.8: The heat distribution in a rod of length $L = 10m$ by finite difference method, using a uniform mesh with 10 points.

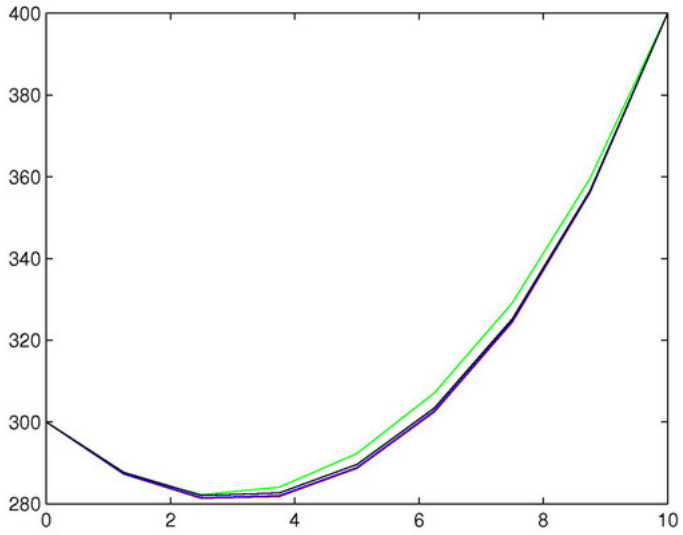

Figure 8.9: The stationary heat distribution in a rod of length $L = 10m$ with different numerical methods on the uniform mesh with the step-size $\Delta x = 1.25$.

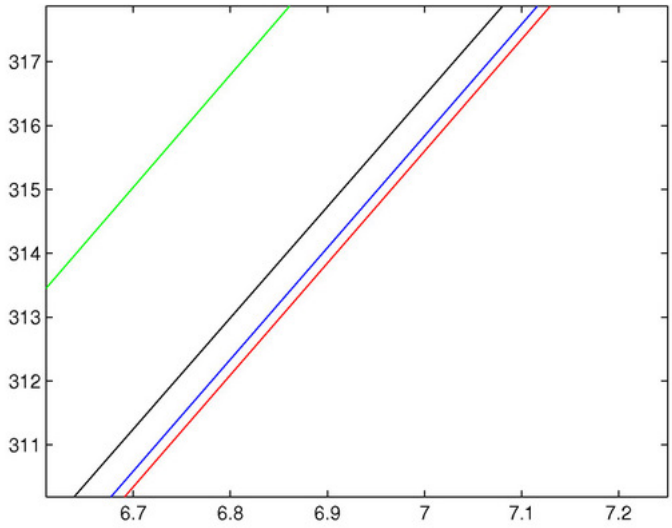

Figure 8.10: Zoom of Figure 8.9. The exact solution is red, the shooting method with the explicit Euler method is green, the shooting method with the improved Euler method is black, the finite difference method is blue.

Bibliography

[1] Asher, U., Petzold, L. Computer Methods for Ordinary Differential Equations, SIAM, Philadelphia, 1998.

[2] Berman, A., Plemmons, R. J. Nonnegative Matrices in the Mathematical Sciences, New York, SIAM, 1987.

[3] Bulirsch, R., Stoer, J. Introduction to Numerical Analysis, Heidelberg, Springer, 1980.

[4] Butcher, J. Numerical Methods for Ordinary Differential Equation, Wiley, Chicester, 2008.

[5] Coddington, E. A., Levinson, N. Theory of differential equations, New York, McGraw-Hill, 1955.

[6] Faragó I., Horváth R. Numerikus mdszerek, Typotech, Budapest, 2011.

[7] Kincaid, D., Cheney, W. Numerical Analysis, Mathematics of Scientific Computing, American Mathematical Society, 2009.

[8] Lambert, J. D. Numerical Methods for Ordinary Differential Systems, Wiley Publ., 2000.

[9] Logan, J. D. A First Course in Differential Equations, Springer, 2006.

[10] Shampine, L. F., Gladwell, I., Thompson, S. Solving ODEs with MATLAB, Cambrigde University Press, Cambridge, 2003.

[11] Simon P., Tóth J. Differencilegyenletek. Bevezets az elmletben s az alkalmazsokba, TypoTech, Budapest, 2005.

[12] Stoyan G., Takó G. Numerikus mdszerek 2., TypoTeX, Budapest, 1995.

[13] Süli, E., Mayers, D. F. An Introduction to Numerical Analysis, Cambridge University Press, 2003.

[14] Varga, R. S. Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.