

Chapter 10

Finite Difference Methods

Finite difference methods were used centuries ago, long before computers were available. As we have seen in Chap. 2, these methods arise quite naturally by going back to the definition of derivatives, and just stopping short of taking the limit as the step size tends to zero. We shall first discuss ordinary differential equations and then partial differential equations.

10.1 Ordinary Differential Equations

Consider an initial value problems of the type

$$\begin{aligned}\frac{du}{dt} &= g(u), \\ u(0) &= f,\end{aligned}\tag{10.1}$$

where $g(u)$ is a general function of the solution u itself, and f is a known value. The simple method (2.9) discussed in Sect. 2.2.2 generalizes to

$$\begin{aligned}u_{n+1} &= u_n + g(u_n)\Delta t, \quad n = 0, 1, \dots, \\ u_0 &= f\end{aligned}$$

for our equation. The method is called the *Euler method* after Leonhard Euler, and it provides a very simple formula for implementation on a computer. When u_n is known, the value $g(u_n)$ and the right hand side can be computed. However, depending on the properties of $g(u)$, it may be necessary to use other forms of discretizations. For example, a better centering of $g(u)$ in the interval $[t_n, t_{n+1}]$ would improve the result. By taking the average of the end points we get the so called *trapezoidal rule*

$$\begin{aligned}u_{n+1} &= u_n + \frac{g(u_n) + g(u_{n+1})}{2}\Delta t, \quad n = 0, 1, \dots, \\ u_0 &= f.\end{aligned}$$

However, there is one complication here. In order to compute u_{n+1} at each step, we must solve a nonlinear equation

$$u_{n+1} - \frac{g(u_{n+1})}{2} \Delta t = u_n + \frac{g(u_n)}{2} \Delta t.$$

When the new time level is involved in this way, we have an *implicit method* in contrast to *explicit methods*. Except for very simple functions $g(u)$, we must use numerical methods for solving the nonlinear equation at each step. This is a typical situation in computational mathematics. In order to obtain more accurate numerical solutions, we may have to design more complicated numerical methods. However, we must make sure that the increased manual effort in construction and programming, results in a faster solution procedure on the computer for obtaining a certain accuracy.

Let us now study the linear problem

$$\begin{aligned} \frac{du}{dt} &= -u, \\ u(0) &= 1. \end{aligned} \tag{10.2}$$

It has the exponentially decreasing solution $u(t) = e^{-t}$, and there is of course no need to use a numerical method. But we do that anyway to illustrate some interesting phenomena. The Euler method is

$$u_{n+1} = u_n - u_n \Delta t, \quad n = 0, 1, \dots,$$

and we try the different time steps $\Delta t = 2.1$ and $\Delta t = 1.9$. The result is shown in Fig. 10.1 together with the true solution. Clearly, the numerical solution is completely wrong. Furthermore, for the larger time step, the amplitude of the solution is growing, and it will never approach zero which the true solution will do for large t .

It is easy to see why things are going wrong. The scheme can be written as

$$u_{n+1} = (1 - \Delta t)u_n, \quad n = 0, 1, \dots$$

The number sequence $\{u_n\}$ is nonincreasing for increasing n if $|1 - \Delta t| \leq 1$, i.e., if $\Delta t \leq 2$. We call the method stable if this condition is satisfied. For $1 < \Delta t \leq 2$ the solution will be oscillating, but at least it will not take off without bounds.

Let us next modify the scheme such that the right hand side $-u$ of the differential equation is taken as $-u_{n+1}$ in the interval $[t_n, t_{n+1}]$. Then the method becomes

$$u_{n+1} = u_n - u_{n+1} \Delta t, \quad n = 0, 1, \dots,$$

or equivalently

$$u_{n+1} = \frac{1}{1 + \Delta t} u_n, \quad n = 0, 1, \dots$$

This is called the *Euler backward* method. We run the same cases as above, and the result is shown Fig. 10.2. The oscillatory behavior is gone and, despite the very large time steps, the solution looks reasonable for all t .

The behavior is again easy to explain. The number multiplying u_n satisfies the stability condition $|1/(1 + \Delta t)| \leq 1$ for all Δt , and we call the method *unconditionally stable*.

Fig. 10.1 Solution of (10.2) by the Euler method, $\Delta t = 2.1$ (---) $\Delta t = 1.9$ (-·-), true solution (—)

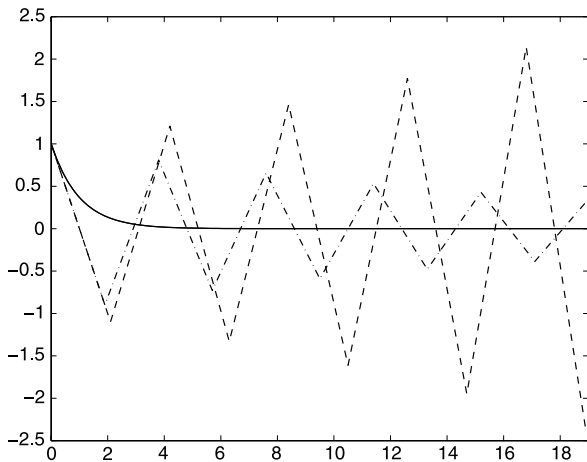
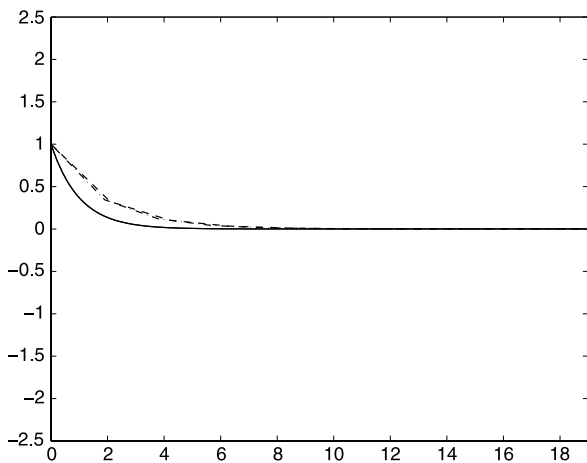


Fig. 10.2 Solution of (10.2) by the Euler backward method, $\Delta t = 2.1$ (---) $\Delta t = 1.9$ (-·-), true solution (—)



It seems that it should not make much difference if we choose to approximate the right hand side $-u$ of (10.2) by the value $-u_n$ at one end of the interval or by $-u_{n+1}$ at the other end. But obviously it does.

For systems of ODE, the methods for scalar equations can be generalized by simply switching to vector notation. For example, the Euler backward method for the differential equation

$$\frac{d\mathbf{u}}{dt} = \mathbf{g}(t, \mathbf{u})$$

is

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \mathbf{g}(t_{n+1}, \mathbf{u}_{n+1})\Delta t.$$

This is simple enough to write down, but what does it take to solve it? The unknown vector is \mathbf{u}_{n+1} , and it is determined by the vector equation

$$\mathbf{u}_{n+1} - \mathbf{g}(t_{n+1}, \mathbf{u}_{n+1})\Delta t = \mathbf{u}_n. \quad (10.3)$$

This is a nonlinear system of N equations for the N unknown elements in \mathbf{u}_{n+1} . It seems like a hopeless task to solve such a system for each time step, but it is not. We shall discuss iterative solution methods for it in Chap. 13.

The analysis of a system of ODE is much harder compared to a scalar ODE, but there are effective tools to simplify the analysis. We take a linear system

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u},$$

where A is an $N \times N$ matrix. Assuming that A has N linearly independent eigenvectors, we let T be the matrix that takes A to diagonal form (see Sect. 3.4):

$$T^{-1}AT = \Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_N).$$

We now multiply the differential equation from the left by T^{-1} . Since T does not depend on t , and $T^{-1}T = I$, the differential equation can be written as

$$\frac{d(T^{-1}\mathbf{u})}{dt} = T^{-1}AT T^{-1}\mathbf{u},$$

or with $\mathbf{v} = T^{-1}\mathbf{u}$.

$$\frac{d\mathbf{v}}{dt} = \Lambda\mathbf{v}.$$

But this is a set of scalar ODE that are independent of each other, and the analysis has become considerably simpler.

As an example we take a case where all the eigenvalues λ_j are real and negative. This means that

$$|v_j(t)| = |e^{\lambda_j t} v_j(0)| \leq |v_j(0)|, \quad j = 1, 2, \dots, N,$$

and obviously we have

$$\|\mathbf{v}(t)\| \leq \|\mathbf{v}(0)\|$$

for the vector norm. For the original ODE we get

$$\|\mathbf{u}(t)\| = \|T\mathbf{v}(t)\| \leq \|T\| \|\mathbf{v}(t)\| \leq \|T\| \|\mathbf{v}(0)\| \leq \|T\| \|T^{-1}\| \|\mathbf{u}(0)\|.$$

For the original system there may be an increase of the norm, but an a priori bound is known, and it is independent of t . The bound

$$\text{cond}(T) = \|T\| \|T^{-1}\|$$

is called the *condition number* of the matrix T , and it is going to show up again when discussing linear systems of algebraic equations in Chap. 14. Here we conclude that a system of ODE becomes sensitive to perturbations, and therefore harder to solve, when the eigenvectors of the coefficient matrix A are almost linearly dependent.

Let us now analyze the Euler backward method for the same system

$$\mathbf{u}_{n+1} = \mathbf{u}_n + A\mathbf{u}_{n+1}\Delta t.$$

After the same type of transformation as we used for the differential equation, we get

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \Lambda\mathbf{v}_{n+1}\Delta t.$$

From the scalar analysis above we know that each component of \mathbf{v} is a nonincreasing sequence for increasing n . Obviously this leads to the inequality

$$\|\mathbf{v}_n\| \leq \|\mathbf{v}_0\|$$

for the vector norm. For the original scheme we get in the same way as for the ODE system

$$\|\mathbf{u}_n\| = \|T\mathbf{v}_n\| \leq \|T\|\|\mathbf{v}_n\| \leq \|T\|\|\mathbf{v}_0\| \leq \|T\|\|T^{-1}\|\|\mathbf{u}_0\| = \text{cond}(T)\|\mathbf{u}_0\|.$$

The bound is identical to the one for the ODE system.

The conclusion from this exercise is that the eigenvalue analysis is very powerful. It shows that when analyzing a certain difference method for a system of ODE, we gain much knowledge by analyzing how it works for a scalar equation

$$\frac{du}{dt} = \lambda u,$$

which goes under the name the *test equation*. In Sect. 6.1 this equation was discussed briefly as a result of a Fourier transformed PDE. The number λ is there the Fourier transform of a differential operator in space, and its location in the complex plane is essential for the properties of the original PDE. The solution is

$$u(t) = e^{\lambda t}u(0),$$

and we note that it is nonincreasing with time if and only if $\text{Re } \lambda \leq 0$.

But λ may as well be the discrete Fourier transform of a difference operator in space, and in that case the solution of the test equation tells something about the semidiscrete approximation.

As another example of discretization in time, we apply the trapezoidal rule

$$u_{n+1} = u_n + \frac{\lambda\Delta t}{2}(u_n + u_{n+1}),$$

or equivalently

$$u_{n+1} = \frac{1 + \lambda\Delta t/2}{1 - \lambda\Delta t/2}u_n.$$

For negative λ , the sequence $\{u_n\}$ is nonincreasing, and we have an unconditionally stable scheme.

The parameters λ and Δt will always occur as $\mu = \lambda\Delta t$ in the right hand side for any consistent one-step scheme for the test equation, and the general form is

$$u_{n+1} = z(\mu)u_n, \tag{10.4}$$

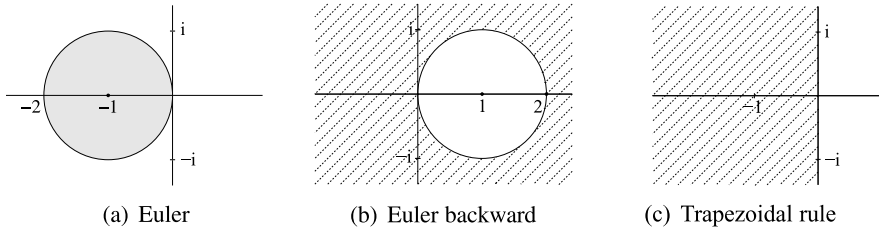


Fig. 10.3 Stability domains

where the *amplification factor* $z(\mu)$ is a scalar function of μ . From a stability point of view the interesting question is for what values of μ do we have $|z(\mu)| \leq 1$. We recall that the eigenvalues of a matrix may be complex even if the matrix is real, and it is therefore necessary to consider complex μ . We make the formal definition:

The *stability domain* for a difference method (10.4) is the set $S(\mu)$ in the complex plane which satisfies $|z(\mu)| \leq 1$.

The (shaded) stability domains for the Euler, Euler backward and trapezoidal method are shown in Fig. 10.3.

Since $\text{Re}(\mu) = \text{Re}(\lambda \Delta t) \leq 0$ if and only if $\text{Re} \lambda \leq 0$, we note that the trapezoidal rule is the only method that is stable for exactly those values of λ where the true solution is nonincreasing. The Euler method has further stability restrictions, while the Euler backward method is “overstable”, i.e., it is stable also for certain λ where the true solution grows.

A few warnings concerning the test equation are appropriate. The assumption of a full set of eigenvectors is not always fulfilled, and then a scalar ODE doesn’t tell it all. Secondly, the matrix A may depend on t , and the diagonalization does not go through that easily.

Even worse is of course a nonlinear ODE. In that case one can linearize the equation, which we shall sketch for the ODE

$$\frac{du}{dt} = g(u), \quad (10.5)$$

where $g(u)$ is a nonlinear function of u . We make a small perturbation $u \rightarrow u + v$, where $|v|$ is small, and plug it into the differential equation. A Taylor expansion gives

$$\frac{du}{dt} + \frac{dv}{dt} = g(u + v) = g(u) + \frac{dg}{du}(u)v + \mathcal{O}(|v|^2).$$

When using the differential equation (10.5) and neglecting the square terms, we get the equation

$$\frac{dv}{dt} = \frac{dg}{du}(u)v.$$

If we now assume that the function $u = u(t)$ is known, we have a linear differential equation for $v(t)$. In a real application we do not of course know u , since that is the solution we want to compute. But we may know for example that the derivative

dg/du is negative, which brings us back to the example above. By understanding how the linear problem behaves, we know how a small perturbation of the solution to the original problem develops with time. In the same way we gain knowledge about the original difference scheme by studying the corresponding linear difference scheme.

The procedure described here is known as *linearization* of the equation, and is a very common analysis tool. If we know that there is a bound on small perturbations when time increases, the computation can be done with more confidence.

A fundamental question is of course how accurate the numerical solution is. As an example, we consider the Euler method. The first question is how well the difference scheme approximates the differential equation, and the answer is obtained by substituting the true solution $u(t)$ of the differential equation into the difference scheme. Since it cannot be expected to satisfy this scheme exactly, we have

$$u(t_{n+1}) = u(t_n) + \Delta t g(u(t_n)) + R,$$

and the question is how big is the remainder R ? The Taylor expansion gives

$$u(t_{n+1}) = u(t_n) + \Delta t \frac{du}{dt}(t_n) + \frac{\Delta t^2}{2} \frac{d^2u}{dt^2}(t_n) + \mathcal{O}(\Delta t^3),$$

and by using the differential equation $du/dt = g$ we get

$$u(t_{n+1}) = u(t_n) + \Delta t g(t_n) + \frac{\Delta t^2}{2} \frac{d^2u}{dt^2}(t_n) + \mathcal{O}(\Delta t^3).$$

Since we are dealing with differential equations, it is natural to normalize the equation by dividing by Δt :

$$\frac{u(t_{n+1}) - u(t_n)}{\Delta t} = g(t_n) + \frac{\Delta t}{2} \frac{d^2u}{dt^2}(t_n) + \mathcal{O}(\Delta t^2).$$

By letting Δt tend to zero, we recover the differential equation in the limit. The error for finite but small Δt

$$T(\Delta t) = \frac{\Delta t}{2} \frac{d^2u}{dt^2}(t_n) + \mathcal{O}(\Delta t^2) = \mathcal{O}(\Delta t)$$

is called the *truncation error*. (The error $R = \mathcal{O}(\Delta t^2)$ defined above is called the *local truncation error*.)

It is important to distinguish between the truncation error on one hand, describing the error in the approximation of the *differential equation*, and the error $u_n - u(t_n)$ in the approximate *solution* on the other hand. It can be shown that they are of the same order under the important condition that the difference scheme is stable in a certain sense. We shall not go into those details here. For linear ODE the analysis is not very difficult. For the Euler and Euler backward schemes one can show that the error is of the order $\mathcal{O}(\Delta t)$, while it is $\mathcal{O}(\Delta t^2)$ for the trapezoidal rule.

In general, if $T(\Delta t) = \mathcal{O}(\Delta t^p)$ with $p > 0$, then the difference scheme is *consistent*, and we say that the difference scheme has *order of accuracy* p . If we also have $|u_n - u(t_n)| = \mathcal{O}(\Delta t^p)$ with $p > 0$, then the numerical solution *converges* to the true solution as $\Delta t \rightarrow 0$, i.e., for any fixed time $t = T$ we have

$$\lim_{\Delta t \rightarrow 0} |u_{T/\Delta t} - u(T)| = 0.$$

We say that the difference scheme is *convergent*.

In practical computations one can of course never reach the limit $\Delta t = 0$. However, the theoretical concept of convergence is still fundamental. If a certain computation gives a result that is not accurate enough, we would like to get a more accurate result if the computation is repeated with a smaller time step. This can be expected with a convergent difference scheme.

The examples we have discussed so far have order of accuracy one or two. The difference methods used in practice are often of higher order. There are essentially two ways of achieving this. One is to aim for *one-step methods* where only one time level t_n is used for computing u_{n+1} . This requires several stages in the computation, and we arrive at the large class of *Runge–Kutta methods*, named after the German mathematicians Carl Runge (1856–1927) and Martin Wilhelm Kutta (1867–1944). The most common method is the fourth order version

$$\begin{aligned} k_1 &= g(u_n), \\ k_2 &= g\left(u_n + \frac{\Delta t}{2}k_1\right), \\ k_3 &= g\left(u_n + \frac{\Delta t}{2}k_2\right), \\ k_4 &= g(u_n + \Delta t k_3), \\ u_{n+1} &= u_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4). \end{aligned}$$

It may seem like a strange formula, but the simple test equation $du/dt = \lambda u$ indicates how it is derived. For this equation we have

$$\frac{du}{dt} = \lambda u, \quad \frac{d^2u}{dt^2} = \lambda^2 u, \quad \frac{d^3u}{dt^3} = \lambda^3 u, \quad \frac{d^4u}{dt^4} = \lambda^4 u,$$

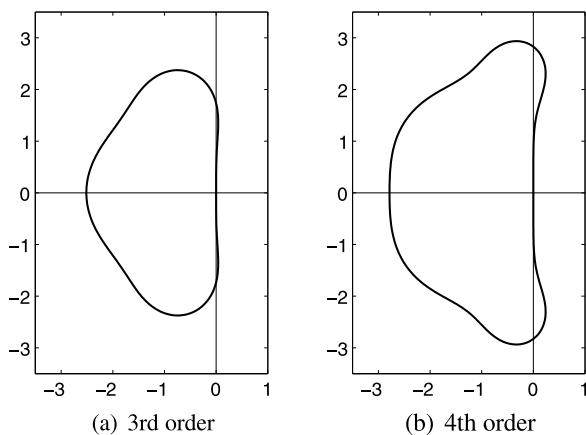
and by Taylor expansion

$$\begin{aligned} u(t + \Delta t) &= u(t) + \Delta t \lambda u(t) + \frac{\Delta t^2}{2} \lambda^2 u(t) \\ &\quad + \frac{\Delta t^3}{6} \lambda^3 u(t) + \frac{\Delta t^4}{24} \lambda^4 u(t) + \mathcal{O}(\Delta t^5). \end{aligned}$$

The Runge–Kutta method for our equation is

$$\begin{aligned} k_1 &= \lambda u_n, \\ k_2 &= \left(\lambda + \frac{\Delta t}{2} \lambda^2\right) u_n, \\ k_3 &= \left(\lambda + \frac{\Delta t}{2} \lambda^2 + \frac{\Delta t^2}{4} \lambda^3\right) u_n, \\ k_4 &= \left(\lambda + \Delta t \lambda^2 + \frac{\Delta t^2}{2} \lambda^3 + \frac{\Delta t^3}{4} \lambda^4\right) u_n, \\ u_{n+1} &= \left(1 + \Delta t \lambda + \frac{\Delta t^2}{2} \lambda^2 + \frac{\Delta t^3}{6} \lambda^3 + \frac{\Delta t^4}{24} \lambda^4\right) u_n, \end{aligned}$$

Fig. 10.4 Stability domains for Runge–Kutta methods



i.e., exactly the Taylor expansion above. After dividing by Δt , we find that the truncation error is $\mathcal{O}(\Delta t^4)$ as it should be. After a little more work for the general nonlinear differential equation, the result is the same. The Runge–Kutta method presented here has fourth order accuracy.

Note that it is a one-step method in the sense that the solution u_n at only one time level is required in order to compute u_{n+1} . But there are several *stages* in the computational procedure.

The stability domain $S(\mu)$ is obtained by finding the values of μ for which

$$|z(\mu)| = \left| 1 + \mu + \frac{\mu^2}{2} + \frac{\mu^3}{6} + \frac{\mu^4}{24} \right| \leq 1.$$

In Fig. 10.4 $S(\mu)$ is shown for both the third and fourth order Runge–Kutta methods. In the third order case, $z(\mu)$ has the same expansion as in the fourth order case, except for the last term, which is not present.

Runge–Kutta type methods of very high order have been derived over the years. They all have the same structure as above, but the number of stages grows with higher order of accuracy (more expressions k_j to be stored).

The method above is explicit, but there are also implicit Runge–Kutta methods. They have the same overall structure but, in the formula for k_j , the same quantity k_j occurs also in the right hand side in the argument v of $g(v)$. This requires the solution of a nonlinear equation at each stage for each step $u_n \rightarrow u_{n+1}$. The advantage is that the stability properties improve.

Another way of constructing high order methods is to involve more than two time levels when advancing the solution one step. The simplest such method is the second order accurate *leap-frog method*

$$u_{n+1} = u_{n-1} + 2\Delta t g(u_n).$$

It is a special case of a *linear multistep method*, and it requires two initial values to get started. If u_0 is a given initial value for the differential equation, we need also u_1 . That value must be computed by a one-step method.

A general linear multistep method has the form

$$\begin{aligned} & \alpha_m u_{n+m} + \alpha_{m-1} u_{n+m-1} + \cdots + \alpha_0 u_n \\ & = \Delta t g(\beta_m u_{n+m} + \beta_{m-1} u_{n+m-1} + \cdots + \beta_0 u_0). \end{aligned}$$

The word “linear” in the name for this class refers to the fact that $g(u)$ occurs in a linear way in the formula (not as $g^2(u)$ for example), and has nothing to do with the type of function $g(u)$, which may very well be nonlinear. One can also use the different form

$$\begin{aligned} & \alpha_m u_{n+m} + \alpha_{m-1} u_{n+m-1} + \cdots + \alpha_0 u_n \\ & = \Delta t (\beta_m g(u_{n+m}) + \beta_{m-1} g(u_{n+m-1}) + \cdots + \beta_0 g(u_0)), \end{aligned}$$

which has the same order of accuracy. There is a significant flexibility in the choice of coefficients α_j and β_j . For a given order of accuracy, there are many ways of choosing the coefficients. If β_{n+m} is nonzero, the method is implicit but, if we want to keep the simpler explicit structure obtained with $\beta_{n+m} = 0$ while keeping the order of accuracy, we have to add more time levels at the other end.

The leap-frog method above has a symmetric and simple structure, and it is tempting to generalize it to higher order. By Taylor expansion it is easy to show that

$$\begin{aligned} \frac{du}{dt}(t) &= \frac{1}{\Delta t} \left(-\frac{1}{12} u(t+2\Delta t) + \frac{2}{3} u(t+\Delta t) - \frac{2}{3} u(t-\Delta t) + \frac{1}{12} u(t-2\Delta t) \right) \\ &+ \mathcal{O}(\Delta t^4), \end{aligned} \tag{10.6}$$

which leads to the simple fourth order method

$$-\frac{1}{12} u_{n+4} + \frac{2}{3} u_{n+3} - \frac{2}{3} u_{n+1} + \frac{1}{12} u_n = \Delta t g(u_{n+2}). \tag{10.7}$$

In order to find out about the stability domain for the test equation, $g(u_{n+2})$ is replaced by λu_{n+2} . It is easy to determine when a given one-step method is stable as we saw above, but here we encounter a new difficulty. When more time levels are involved, how do we analyze stability? We write a general difference equation as

$$c_m u_{n+m} + c_{m-1} u_{n+m-1} + \cdots + c_0 u_n = 0.$$

The key to the analysis is the roots z_j of the *characteristic equation*

$$c_m z^m + c_{m-1} z^{m-1} + \cdots + c_0 = 0,$$

which is formally obtained by substituting $u_n = z^n$ and then dividing by z^n . If all the roots are distinct, then the general solution has the form

$$u_n = a_1 z_1^n + a_2 z_2^n + \cdots + a_m z_m^n,$$

where the constants a_j are determined by the m initial conditions. For stability we require that the solution has no growing component, and obviously the condition is $|z_j| \leq 1$ for all j . If there is a double root z_1 , the form of the solution is

$$u_n = (a_1 + a_2 n) z_1^n + a_3 z_3^n + \cdots + a_m z_m^n.$$

If $|z_1| = 1$, the solution will grow without bound when n increases. If on the other hand $|z_1| < 1$, then the component $a_2 n z_1^n$ will grow initially as n increases, but then it will decrease. This means that the solution stays bounded by a constant K which is independent of n . If there is a root with multiplicity higher than two, the polynomial multiplying it will be of higher degree, but the conclusion is again that the solution stays bounded independent of n if $|z_1| < 1$.

For the test equation, the roots will be functions of $\mu = \lambda \Delta t$. The definition of the stability domain for linear multistep methods is:

$$S = \{\mu : \text{all roots } z_j(\mu) \text{ satisfy } |z_j(\mu)| \leq 1, \text{ multiple roots satisfy } |z_j(\mu)| < 1\}.$$

Let us now go back to the leap-frog method. The characteristic equation is

$$z^2 - 2\mu z - 1 = 0$$

with the roots

$$z_{1,2} = \mu \pm \sqrt{\mu^2 + 1}.$$

It is easily shown that the stability domain is just the line segment

$$\{\mu : \operatorname{Re} \mu = 0, |\operatorname{Im} \mu| < 1\}$$

on the imaginary axis. However, it is not as bad as it looks. Many problems are such that the coefficient matrix of the linearized system of ODE has purely imaginary eigenvalues. A simple example is

$$\frac{du}{dt} = v, \quad \frac{dv}{dt} = -u,$$

which can be written as

$$\frac{d\mathbf{u}}{dt} = A\mathbf{u}, \quad \mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad (10.8)$$

The eigenvalues of A are given by

$$\lambda^2 + 1 = 0,$$

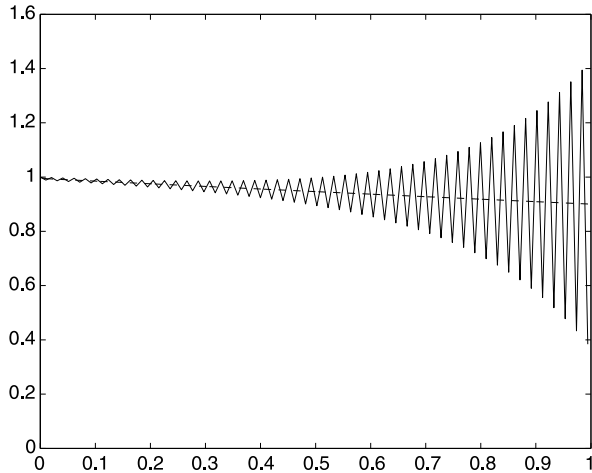
i.e., $\lambda_{1,2} = \pm i$. Accordingly, the leap-frog scheme

$$\mathbf{u}_{n+1} = \mathbf{u}_{n-1} + 2\Delta t A \mathbf{u}_n.$$

is stable for $\Delta t < 1$.

Next we take a look at the fourth order method (10.7) derived above. A minimal requirement is that it should be stable for $\mu = 0$ corresponding to a constant solution u independent of time. For the fourth order method above, it turns out that one of the roots is $z_1 = 7.873$ showing that the method is completely useless. The same conclusion follows for any symmetric method of the same type with accuracy 6, 8, \dots . Here we have a case where increasing the formal order of accuracy has a negative effect. The stability is a key concept that always must be kept in mind.

Fig. 10.5 Solution of (10.9) by the Euler method, $\Delta t = 0.001$ (---), $\Delta t = 0.01025$ (—)



Finally we shall discuss an important class of ODE that is quite common in applications. Consider the initial value problem

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} u \\ v \end{bmatrix} &= \begin{bmatrix} -100 & 99.9 \\ 99.9 & -100 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}, \\ u(0) &= 1, \\ v(0) &= 0.99. \end{aligned} \tag{10.9}$$

We apply the Euler method with the two different time-steps $\Delta t = 0.001$ and $\Delta t = 0.01025$, and the result is shown in Fig. 10.5 for $u(t)$. The dashed curve with the shorter time step is smooth, and as we will see below, it is close to the true solution. The solution with the larger time step goes completely wrong. Obviously we have again a case with an instability, but there is actually a new observation to be made here.

We go back to the scalar problem (10.2) with the true solution shown in Fig. 10.2. In this case the stability limit is not a severe restriction, since the time steps have to be relatively small anyway in order to resolve the solution properly at the beginning of the time interval. Actually, a time step of the order $\Delta t = 0.1$ seems quite reasonable when looking at the graph, and this is far below the stability limit $\Delta t = 2$. The solution of (10.9) is very smooth, and a time step of the order $\Delta t = 0.1$ is certainly enough to get a good resolution. What is the reason for the severe time restriction?

The coefficient matrix has the two eigenvalues $\lambda_1 = -0.1$ and $\lambda_2 = -199.9$, i.e., they are far apart. Such a system of ODE is called *stiff*, and it is very common in many different types of applications. One of them occurs in chemistry when dealing with a number of components where the chemical reactions take place on different time scales. Another application is obtained when discretizing partial differential equations as we shall see later on.

With the new variables $\phi = (u + v)/2$ and $\psi = (u - v)/2$ in (10.9) we get

$$\begin{aligned}\frac{d\phi}{dt} &= -0.1\phi, & \frac{d\psi}{dt} &= -199.9\psi, \\ \phi(0) &= 0.995, & \psi(0) &= 0.005,\end{aligned}$$

which is a direct way of diagonalizing the system. The function $\psi(t) = 0.005e^{-199.9t}$ is almost zero all the time, and the other function $\phi(t) = 0.995e^{-0.1t}$ is very smooth. By going back to the original variables $u = \phi + \psi$ and $v = \phi - \psi$, we see that they are very smooth as well. However, the crucial point is that the eigenvalue $\lambda_2 \approx -200$, entering in the form of e^{-200t} , is present all the time, even if it is annihilated for the true solution by the choice of initial values. The discrete solution introduces perturbations triggering the “parasitic” solution, which cannot be handled by the Euler scheme if the time steps are not extremely small.

In the example above, a good approximation would be to assume from the beginning that $u - v = 0$ all the time. For the general case, such a type of assumption leads to a new type of system. Let $\mathbf{u}(t)$ and $\mathbf{v}(t)$ be two vector functions with m and n components respectively. A general *differential-algebraic system* has the form

$$\begin{aligned}\frac{d\mathbf{u}}{dt} &= \mathbf{f}(\mathbf{u}, \mathbf{v}), \\ \mathbf{g}(\mathbf{u}, \mathbf{v}) &= \mathbf{0}.\end{aligned}$$

Here the vector \mathbf{f} has m components, and \mathbf{g} has n components.

Differential-algebraic systems are limits of stiff systems and can be handled as such when solving them numerically, but in general special methods are used.

Stiff systems and differential-algebraic systems have been studied extensively, and very effective numerical methods are available today.

Modern variants of finite difference methods work with variable step size and even variable order of accuracy. The algorithm contains various types of sensors that are estimating the error. If a local fast change of the solution occurs, the solver works hard to reduce the time step in order to produce a well resolved solution.

There are many ODE-solvers on the market today. The MATLAB system has at least seven solvers, mainly divided into stiff and nonstiff classes. The *ode45*-function is the standard solver, using a Runge–Kutta method.

Exercise 10.1 When analyzing the linear ODE $du/dt = \lambda(t)u$, it is of interest to know the limit $\max_t |\lambda(t)|$. Derive the linearized form of the nonlinear ODE

$$\frac{du}{dt} = \frac{1}{u^2 + 1}, \quad (10.10)$$

and determine the limit of the coefficient that corresponds to $\lambda(t)$.

Exercise 10.2 Consider the initial value problem (10.2) with the solution $u(t) = e^{-t}$. Prove that the difference scheme

$$u_{n+1} = u_{n-1} - 2\Delta t u_n$$

is unstable for any Δt , while

$$u_{n+1} = u_{n-1} - \Delta t(u_{n+1} + u_{n-1})$$

is stable for all Δt .

Exercise 10.3 Use the MATLAB ODE-solvers *ode45* and *ode23* (see Sect. 18.2) for solving (10.10) with the initial value $u(0) = 0$. Compare the choice of time steps for the two methods.

10.2 Partial Differential Equations

Let us now turn to partial differential equations and the so-called transport equation (2.18) with $c = 1$. We introduce the two-dimensional grid $(x_j, t_n) = (j\Delta x, n\Delta t)$ and the grid function u_j^n as an approximation of $u(j\Delta x, n\Delta t)$, see Fig. 10.6.

Note that we have switched notation from the previous section by changing the subscript n indicating time level t_n , to a superscript. In this way it is easier to distinguish from the subscript j indicating the grid point in space. One has to be careful though, not to confuse the superscript n with the power notation.

At a certain point (x_j, t_n) we substitute

$$\begin{aligned}\frac{\partial u}{\partial x} &\rightarrow \frac{u_{j+1}^n - u_j^n}{\Delta x}, \\ \frac{\partial u}{\partial t} &\rightarrow \frac{u_j^{n+1} - u_j^n}{\Delta t}\end{aligned}$$

in the differential equation and obtain

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(u_{j+1}^n - u_j^n). \quad (10.11)$$

If the initial function $u_j^0 = u(x_j, 0)$ is known, we can compute u_j^1 for all j , then u_j^2 and so on.

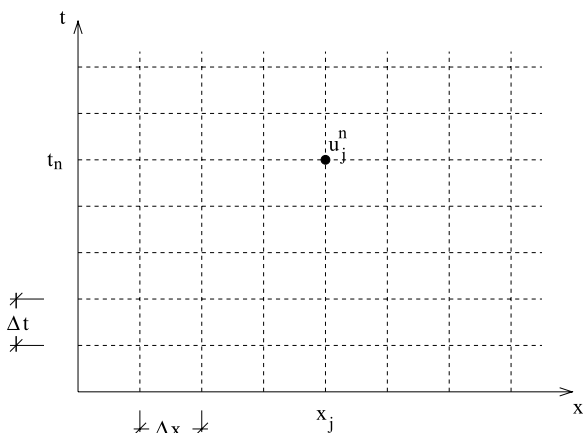
Let us now try an experiment. An initial pulse is defined as in Sect. 9.2 with its center at $x = 0.2$ as

$$u(x, 0) = e^{-800(x-0.2)^2}.$$

The solution at time t is

$$u(x, t) = e^{-800(x-t-0.2)^2},$$

i.e., the pulse has moved a distance t to the right. We do the computation in the x -interval $[0, 1]$ with $N = 1/\Delta x$ grid points, and choose $\Delta t = 0.8\Delta x$. Figure 10.7(a) shows the result at $t = 0.038$ for $N = 400$. It is centered properly at $x = 0.238$, but the peak is too high. It is reasonable to assume that a finer grid should give better results, since the difference scheme approximates the differential equation more closely. However, we are in for a surprise. With half the step size in both directions,

Fig. 10.6 Computational grid

we get the result shown in Fig. 10.7(b). The solution goes completely wrong and shows strange oscillations. What has happened?

We have defined the derivative as a limit of a *forward difference* $(u(x + \Delta x, t) - u(x, t))/\Delta x$. As noted earlier it is of course also possible to define it as a limit of a *backward difference* $(u(x, t) - u(x - \Delta x, t))/\Delta x$. When using this as the basis for our difference approximation, we get

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(u_j^n - u_{j-1}^n). \quad (10.12)$$

It turns out that the numerical solution now behaves well, and we can compute it over long time. Figure 10.8 shows the result at $t = 0.4$ with the same data as above. For $N = 400$ the pulse is centered at the right position $x = 0.6$, but the top is too low. With half the step size we get a better result as shown in Fig. 10.8(b).

The first computation with forward differences in the approximation is an example of an *unstable* computation, while the second one with *backward differences* is a *stable* computation. Actually, there is a simple explanation for the bad behavior of the first one. At any given grid point (x_j, t_{n+1}) the approximation doesn't use any point to the left at the previous time level. Since the pulse is moving to the right, we must know what is coming in from the left. The second approximation takes this into account.

Actually, there is not plain sailing with the second approximation either. We do the same computation with $\Delta t = 1.2\Delta x$, and the result is shown in Fig. 10.9 at $t = 0.32$. Severe oscillations have occurred, even quite far from the pulse, and obviously the numerical solution is useless. Apparently, the time step must be chosen small enough in order to retain the stability. This is in accordance with the discussion about stability domains for ODE. For PDE the theoretical analysis is harder but, by using Fourier analysis to be presented in the next section, one can show that stability requires the condition $\Delta t \leq \Delta x$.

Let us next compute the solution to the heat conduction problem (2.20). We construct a difference scheme by using the three point expression used in the definition of the second derivative $\partial^2 u / \partial x^2$ and a forward finite difference for $\partial u / \partial t$. With

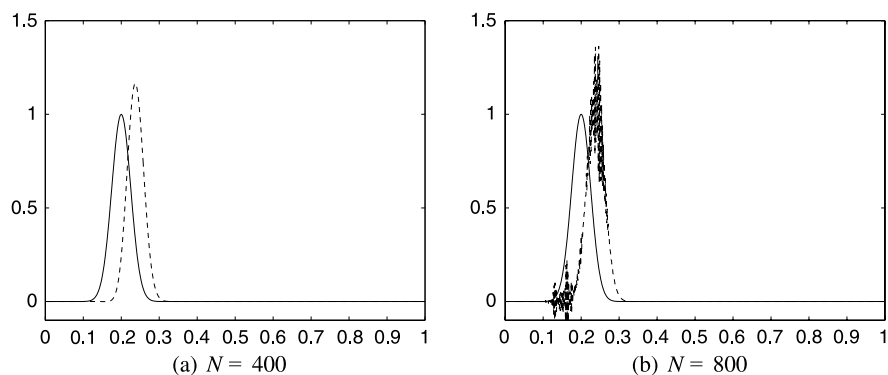


Fig. 10.7 Solution of (10.11), $t = 0$ (—) and $t = 0.038$ (---)

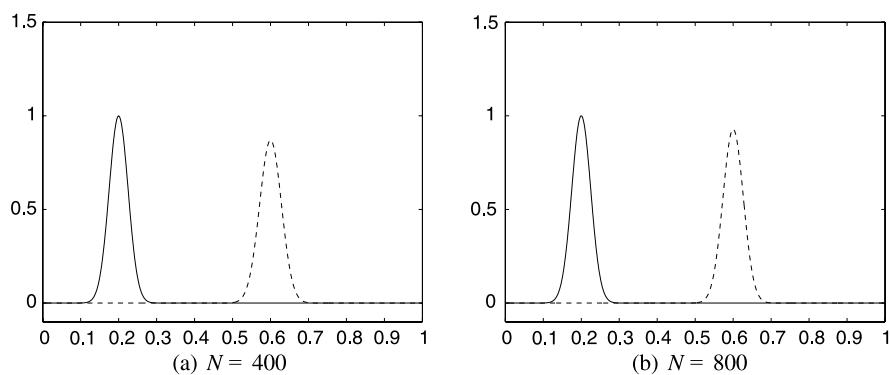
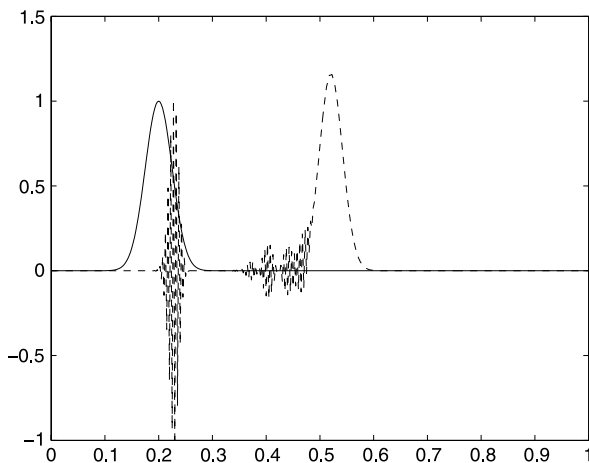


Fig. 10.8 Solution of (10.12) with $\Delta t = 0.8\Delta x$, $t = 0$ (—) and $t = 0.4$ (---)

Fig. 10.9 Solution of (10.12) with $\Delta t = 1.2\Delta x$, $t = 0$ (—) and $t = 0.32$ (---)



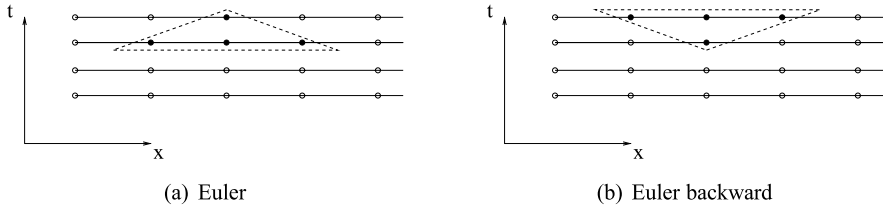


Fig. 10.10 Computational stencils for the heat equation

$N + 1$ grid points in the x -direction including the boundary points, the resulting scheme is

$$\begin{aligned}
 u_j^{n+1} &= u_j^n + \frac{\Delta t}{\Delta x^2} (u_{j-1}^n - 2u_j^n + u_{j+1}^n), \\
 j &= 1, 2, \dots, N-1, \quad n = 0, 1, \dots, \\
 u_0^{n+1} &= 1, \\
 u_N^{n+1} &= 1, \\
 u_j^0 &= f_1(x_j),
 \end{aligned}$$

as illustrated in Fig. 10.10(a). With u_j^0 known for all j , u_j^1 can be computed for all j , and so on until we reach the final time level.

Even if the solution is obtained by stepping forward in time, one could still use an approximation based on the Euler backward scheme, just as for ordinary differential equations discussed above. The forward difference in time is replaced by a backward difference, which gives

$$\begin{aligned}
 u_j^{n+1} &= u_j^n + \frac{\Delta t}{\Delta x^2} (u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}), \\
 j &= 1, 2, \dots, N-1, \quad n = 0, 1, \dots, \\
 u_0^{n+1} &= 1, \\
 u_N^{n+1} &= 1, \\
 u_j^0 &= f_1(x_j),
 \end{aligned} \tag{10.13}$$

see Fig. 10.10(b). This is an implicit scheme, and it requires more computation for each time step. Since all grid points at the new time level t_{n+1} are coupled to each other, this complication is more severe compared to ODE. We must solve a large system of equations for each time step.

For a well posed problem, there is still a possibility that the numerical scheme goes wrong as we saw for the equation $\partial u / \partial t + \partial u / \partial x = 0$ above. We run the first explicit approximation above with two different time steps. Figure 10.11 shows the result for the step size $\Delta t = 0.000310$ and $\Delta t = 0.000315$.

Apparently there is a critical limit somewhere in between these two values. Indeed, by using analytical tools the theoretical stability limit on Δt can be found. In the next section we shall describe how this can be done.

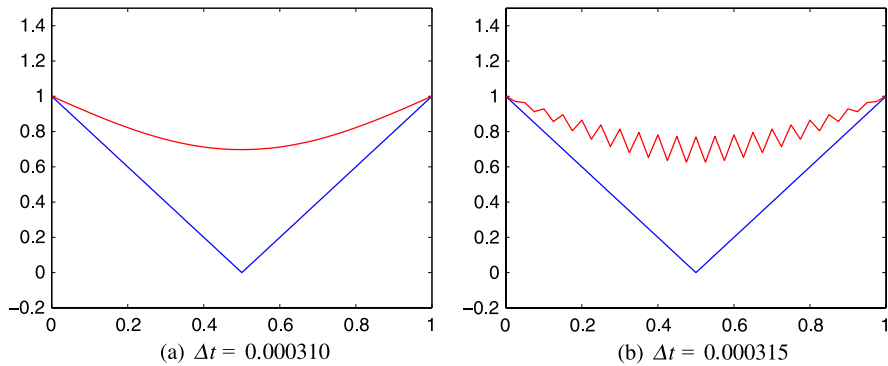


Fig. 10.11 The heat equation

The implicit scheme above does not have any stability restriction on the time step. It is *unconditionally stable*, which is typical for implicit schemes.

The consistency and convergence concepts can be defined in analogy with ordinary differential equations. A *consistent* difference scheme approaches formally the differential equation as $\Delta x \rightarrow 0$ and $\Delta t \rightarrow 0$. It is *convergent* if for any fixed $t = T$ the numerical solution converges to the true solution:

$$\|u_j^{T/\Delta t} - u(x_j, T)\| \rightarrow 0 \quad \text{as } \Delta x \rightarrow 0, \Delta t \rightarrow 0.$$

The norm is a measure of the discrete function at a given time level corresponding to vector norms.

The consistency is usually easy to verify by a direct application of Taylor expansions, but convergence is not. However, there is a fundamental theorem saying that a consistent scheme is convergent if it is stable. Therefore, stability analysis is the key to the construction of accurate difference schemes. In the next section we shall indicate how it can be carried out by using a Fourier technique.

Exercise 10.4 Show that the difference method (10.13) requires the solution of a tridiagonal system (see Sect. 3.3) for each time step. Write down the system in detail.

Exercise 10.5 Suggest a difference method for $\partial u / \partial t = \partial^2 u / \partial x^2$ that uses a combination of $(u_{j-1}^n - 2u_j^n + u_{j+1}^n) / \Delta x^2$ and $(u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}) / \Delta x^2$ for approximation of $\partial^2 u / \partial x^2$.

10.3 Fourier Stability Analysis

In this section we shall describe how one can analyze a given difference scheme for a partial differential equation with respect to its stability properties. We shall limit ourselves to the simplest form of analysis, which is based on the Fourier transform.

Even if the solutions are nonperiodic, it turns out that the analysis of a modified problem with periodic solutions gives significant information about the stability. Let us discuss the solution of the heat equation above, and the difference scheme

$$\begin{aligned} u_j^{n+1} &= u_j^n + \frac{\Delta t}{\Delta x^2} (u_{j-1}^n - 2u_j^n + u_{j+1}^n), \\ j &= 0, 1, \dots, N, \quad n = 0, 1, \dots, \\ u_j^0 &= f_j. \end{aligned} \quad (10.14)$$

Here we have canceled the boundary conditions, and assume periodicity instead: $u_{j+N+1}^n = u_j^n$. For difference methods, only the grid values are accounted for. Therefore, the solution at any time level t_n can be represented as a discrete Fourier series as was demonstrated in Sect. 6.2. We write the series in the form

$$u_j^n = \sum_{k=-N/2}^{N/2} c_k^n e^{ikx_j}, \quad j = 0, 1, \dots, N,$$

where the coefficients are defined by

$$c_k^n = \frac{1}{2\pi} \sum_{j=0}^N u_j^n e^{-ikx_j} \Delta x.$$

The coefficients c_k^n are now time dependent, and the idea is to investigate how these coefficients are behaving when time increases.

At a first glance, it seems like a complication to study these coefficients instead of the original grid values u_j^n . But there are two facts that show why it is a good idea:

1. The difference scheme takes a particularly simple form when it is formulated in terms of the Fourier coefficients.
2. The behavior of the Fourier coefficients is directly related to the behavior of the original grid values via the discrete Parseval's relation (6.10).

We introduce the Fourier series into the difference approximation of the second space derivative and obtain

$$\begin{aligned} u_{j-1}^n - 2u_j^n + u_{j+1}^n &= \sum_{k=-N/2}^{N/2} c_k^n (e^{ikx_{j-1}} - 2e^{ikx_j} + e^{ikx_{j+1}}) \\ &= \sum_{k=-N/2}^{N/2} c_k^n e^{ikx_j} q(\xi), \end{aligned}$$

where

$$q(\xi) = e^{-i\xi} - 2 + e^{i\xi}, \quad \xi = k\Delta x.$$

The whole difference scheme can now be written as

$$\sum_{k=-N/2}^{N/2} \left(c_k^{n+1} - (1 + \sigma q(\xi)) c_k^n \right) e^{ikx_j} = 0,$$

where $\sigma = \Delta t / \Delta x^2$. In Sect. 6.2 it was demonstrated that the $N + 1$ grid functions e^{ikx_j} (which can also be considered as vectors) are linearly independent. By definition this means that each one of the coefficients in the sum must be zero, i.e.,

$$c_k^{n+1} = (1 + \sigma q(\xi)) c_k^n, \quad k = 0, 1, \dots, N, \quad |\xi| \leq \pi. \quad (10.15)$$

This is quite a simplification! The original difference scheme couples neighboring points in space to each other, and the whole set of variables must be treated together. On the contrary, there is no coupling between the Fourier coefficients for different k -values, and the development with time can be handled separately for each one of them. This is in exact analogy with the “continuous” Fourier transform and differential operators as discussed in Sect. 6.1. A differential operator is replaced by multiplication by a number by using the Fourier transform also in that case.

Knowledge about the behavior of the Fourier coefficients is transferred to the solution of the difference scheme by the discrete Parseval’s relation. If we can make sure that the Fourier coefficients do not grow with time, i.e.,

$$|c_k^{n+1}| \leq |c_k^n|, \quad k = 0, 1, \dots, N,$$

then

$$\begin{aligned} \sum_{j=0}^N |u_j^{n+1}|^2 \Delta x &= 2\pi \sum_{k=-N/2}^{N/2} |c_k^{n+1}|^2 \leq 2\pi \sum_{k=-N/2}^{N/2} |c_k^n|^2 \\ &= \sum_{j=0}^N |u_j^n|^2 \Delta x \leq \dots \leq \sum_{j=0}^N |u_j^0|^2 \Delta x. \end{aligned}$$

If we order the grid values u_j^n in a vector \mathbf{u}^n , then the norm is defined by

$$\|\mathbf{u}^n\|^2 = \sum_{j=0}^N |u_j^n|^2 \Delta x,$$

and we have

$$\|\mathbf{u}^n\|^2 \leq \|\mathbf{u}^0\|^2.$$

This could be used as the definition of stability. However, a more reasonable definition is to allow a constant in the estimate:

A difference approximation is stable if the solution satisfies

$$\|\mathbf{u}^n\| \leq K \|\mathbf{u}^0\|,$$

where K is a constant independent of n and \mathbf{u}^0 .

We are now in a very good position. By making sure that the Fourier coefficients satisfy the *von Neumann condition*

$$|c_k^{n+1}| \leq |c_k^n|,$$

we have a final stability estimate for the solution. But (10.15) shows that this condition is satisfied if

$$|1 + \sigma q(\xi)| \leq 1.$$

By using the trigonometric interpretation of $e^{i\xi}$, we get

$$q(\xi) = e^{-i\xi} - 2 + e^{i\xi} = 2 \cos \xi - 2,$$

leading to the inequality

$$|1 - 2\sigma(1 - \cos \xi)| \leq 1.$$

Since $\cos \xi$ never exceeds one, the critical point is $\cos \xi = -1$. This leads to the final condition

$$\sigma = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}.$$

The difference scheme is stable if the time step is chosen small enough:

$$\Delta t \leq \frac{\Delta x^2}{2}.$$

It may seem that the periodicity assumption on the solutions is too restrictive, making the stability result of little value. But it is not. It is actually a necessary stability condition, and it is often sufficient as well. The heat conduction problem above with the temperature specified at both boundaries is such an example. Furthermore, if the heat conduction coefficient depends on x and t , so that the differential equation is

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x} \left(a(x, t) \frac{\partial u}{\partial x} \right)$$

with $a(x, t) > 0$, then the corresponding generalized difference scheme has the stability limit

$$\Delta t \leq \frac{\Delta x^2}{2 \max_{x,t} a(x, t)}.$$

Let us take another look at the transformation procedure used above. If the grid functions are organized as vectors

$$\mathbf{u}^n = \begin{bmatrix} u_0^n \\ u_1^n \\ \vdots \\ u_N^n \end{bmatrix},$$

then we can consider the difference scheme as a relation between the two vectors \mathbf{u}^n and \mathbf{u}^{n+1} connected by a matrix Q :

$$\mathbf{u}^{n+1} = Q\mathbf{u}^n.$$

For our example with periodic solutions, the matrix is

$$Q = \begin{bmatrix} 1-2\sigma & \sigma & & & & \sigma \\ \sigma & 1-2\sigma & \sigma & & & \\ & \sigma & 1-2\sigma & \sigma & & \\ & & \ddots & \ddots & \ddots & \\ & & & \sigma & 1-2\sigma & \sigma \\ \sigma & & & & \sigma & 1-2\sigma \end{bmatrix}.$$

In Sect. 6.2 we introduced the matrix F for the Fourier transform, and we multiply with it from the left, getting

$$F\mathbf{u}^{n+1} = FQF^{-1}F\mathbf{u}^n.$$

With the new vector $\mathbf{v}^n = F\mathbf{u}^n$ and $\Lambda = FQF^{-1}$, we get

$$\mathbf{v}^{n+1} = \Lambda\mathbf{v}^n.$$

The vector \mathbf{v}^n is the Fourier transform of \mathbf{u}^n , and it has the elements c_k^n . Consequently we have a system of equations for each one of the Fourier coefficients c_k^n , and by (10.15) we can see that $\Lambda = FQF^{-1}$ is a diagonal matrix:

$$\Lambda = \text{diag}\left(1 - \sigma q\left(\left(-\frac{N}{2}\right)\Delta x\right), 1 - \sigma q\left(\left(-\frac{N}{2} + 1\right)\Delta x\right), \dots, 1 - \sigma q\left(\frac{N}{2}\Delta x\right)\right).$$

By these arguments we have shown that the application of the Fourier transform is equivalent to diagonalizing the corresponding vector/matrix formulation of the scheme. It is then very easy to do the analysis, since we are now dealing with a set of scalar equations.

For more general initial-boundary value problems, a different kind of theory is required. However, the Fourier type stability analysis is still very powerful. Indeed it is often the only type of analysis that is done for many realistic application problems, and quite often it leads to the correct stability limit on Δt .

Exercise 10.6 Consider the PDE $\partial u / \partial t = \partial^2 u / \partial x^2$ with periodic boundary conditions. Prove that the ODE system that is obtained by discretizing in space by using the standard second order difference operator is stiff (see definition in Sect. 10.1).

Exercise 10.7 Write down the Euler backward difference scheme corresponding to (10.13), but now for the periodic case. Derive the exact form of the system of equations that must be solved for advancing this scheme one step. Compare the form to the nonperiodic case (Exercise 10.4).

Exercise 10.8 Use the Fourier method to prove that the Euler backward method in Exercise 10.7 is unconditionally stable.

Exercise 10.9 Consider the PDE $\partial u / \partial t = a \partial u / \partial x$ and the leap-frog difference scheme

$$u_j^{n+1} = u_j^{n-1} + a \frac{\Delta t}{\Delta x} (u_{j+1}^n - u_{j-1}^n).$$

Use Fourier analysis to derive the stability condition.

10.4 Several Space Dimensions

Problems in one space dimension are almost exclusively used as model problems for analysis and preliminary investigations. Real life problems have almost always at least two space dimensions, and we shall make a few comments on these.

The differential equation has the form

$$\frac{\partial u}{\partial t} = \mathcal{P}(\partial x, \partial y)u$$

with proper initial and boundary conditions. Finite difference methods are not well suited for problems where the computational domain is irregular, since both the construction of the computational grid and the analysis become more complicated. However, for regular geometries, we can use *structured grids*, and the easiest 2D-case is a rectangle $0 \leq x \leq a$, $0 \leq y \leq b$. The grid is defined by

$$(x_{j_1}, y_{j_2}) = (j_1 \Delta x, j_2 \Delta y), \quad j_1 = 0, 1, \dots, N_1, \quad N_1 \Delta x = a, \\ j_2 = 0, 1, \dots, N_2, \quad N_2 \Delta y = b,$$

in the x, y -plane, see Fig. 10.12.

The solution $u(x_{j_1}, y_{j_2}, t_n)$ is approximated by $u_{j_1 j_2}^n$. The Fourier analysis is easily generalized from 1D. The grid function is transformed as

$$u_{j_1 j_2}^n = \sum_{k_1=-N_1/2}^{N_1/2} \sum_{k_2=-N_2/2}^{N_2/2} c_{k_1 k_2}^n e^{i(k_1 x_{j_1} + k_2 y_{j_2})}, \\ j_1 = 0, 1, \dots, N_1, \quad j_2 = 0, 1, \dots, N_2,$$

where the coefficients are defined by

$$c_{k_1 k_2}^n = \frac{1}{(2\pi)^2} \sum_{j_1=0}^{N_1} \sum_{j_2=0}^{N_2} u_{j_1 j_2}^n e^{-i(k_1 x_{j_1} + k_2 y_{j_2})} \Delta x \Delta y.$$

The wave numbers k_1 and k_2 correspond to the wave number k in 1D. After discretization in time and Fourier transformation of the difference scheme in space, we get a number of scalar relations of the type

$$v^{n+1} = q(\xi, \eta) v^n, \quad 0 \leq |\xi|, |\eta| \leq \pi,$$

where $\xi = k_1 \Delta x$, $\eta = k_2 \Delta y$. Also in the 2D-case we have obtained a number of simple algebraic equations instead of a difficult partial differential equation. We simply have to make sure that the amplification factor satisfies the inequality $|q(\xi, \eta)| \leq 1$ for $0 \leq \xi, \eta < 2\pi$, making the difference scheme stable.

Difference schemes can be used for other computational domains than rectangles. As long as we can map the domain to a rectangle we are in good shape. For example, if the boundaries are circular, we use the well known polar coordinates r and θ defined by

$$x = r \cos \theta, \quad y = r \sin \theta.$$

Fig. 10.12 Two-dimensional grid

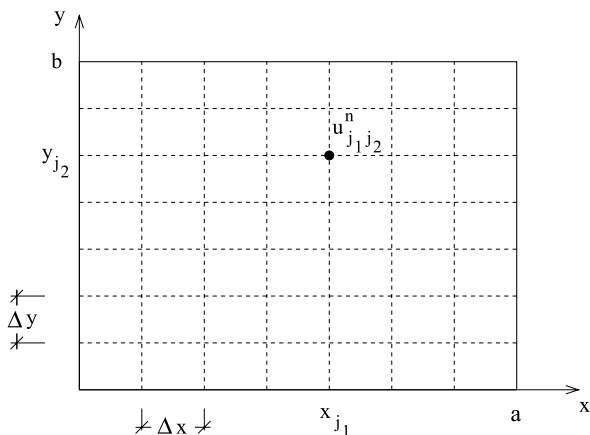


Fig. 10.13 Mapping by using polar coordinates

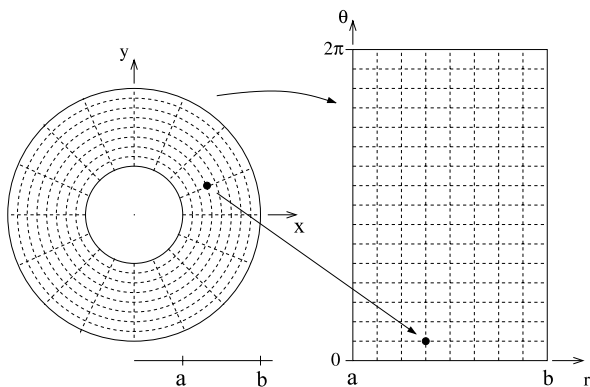


Figure 10.13 shows the mapping for a domain between two circles with radius a and b respectively (the scales are changed in the right figure). The computation is done on the computational grid in the r, θ -plane to the right.

When changing the coordinate system, we must also change the dependent variables and the differential equation. If $u(x, y)$ is a given function, then we get a new function by

$$u(x, y) \rightarrow u(r \cos \theta, r \sin \theta) \rightarrow v(r, \theta).$$

The new differential equation is obtained by using the relations

$$\begin{aligned} \frac{\partial v}{\partial r} &= \frac{\partial u}{\partial x} \frac{\partial x}{\partial r} + \frac{\partial u}{\partial y} \frac{\partial y}{\partial r} = \cos \theta \frac{\partial u}{\partial x} + \sin \theta \frac{\partial u}{\partial y}, \\ \frac{\partial v}{\partial \theta} &= \frac{\partial u}{\partial x} \frac{\partial x}{\partial \theta} + \frac{\partial u}{\partial y} \frac{\partial y}{\partial \theta} = -r \sin \theta \frac{\partial u}{\partial x} + r \cos \theta \frac{\partial u}{\partial y}, \end{aligned}$$

leading to

$$\begin{aligned}\frac{\partial u}{\partial x} &= \frac{1}{r} \left(r \cos \theta \frac{\partial v}{\partial r} - \sin \theta \frac{\partial v}{\partial \theta} \right), \\ \frac{\partial u}{\partial y} &= \frac{1}{r} \left(r \sin \theta \frac{\partial v}{\partial r} + \cos \theta \frac{\partial v}{\partial \theta} \right).\end{aligned}$$

These relations can be further differentiated to obtain higher order derivatives in the new coordinates.

In Sect. 2.3 we introduced the gradient of a function. We must be careful when transferring this concept to a new coordinate system. The direct translation $[v_r, v_\theta]^T$ doesn't work. The definition of the gradient is that it is the vector pointing in the direction where the function has the strongest growth, and with a magnitude that equals this growth rate. Then we must take into account the geometric properties of the new system, and for polar coordinates it turns out that the gradient is

$$\nabla v(r, \theta) = \begin{bmatrix} \frac{\partial v}{\partial r}(r, \theta) \\ \frac{1}{r} \frac{\partial v}{\partial \theta}(r, \theta) \end{bmatrix}.$$

We now go back to the heat equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \quad (10.16)$$

and change variables such that

$$v(r, \theta, t) = u(x(r, \theta), y(r, \theta), t).$$

Then it can be shown that the equation takes the form

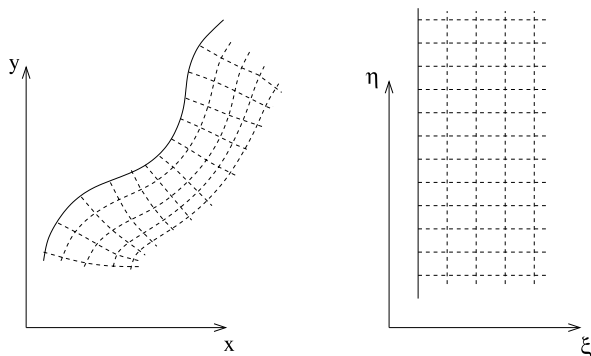
$$\frac{\partial v}{\partial t} = \frac{\partial^2}{\partial r^2} + \frac{1}{r} \frac{\partial}{\partial r} + \frac{1}{r^2} \frac{\partial^2}{\partial \theta^2}.$$

The boundaries would be difficult to represent by a rectangular grid in the original (x, y) -coordinates, and the mapping makes it possible to represent the boundaries exactly while still keeping a structured grid.

There is a technical difficulty with these coordinates if the computational domain contains the center point $r = 0$, since the coefficients $1/r$ and $1/r^2$ become infinite there. Since the physics doesn't know anything about coordinate systems, this singularity has to be an artificial effect caused by the choice of coordinates. One can avoid the problem by excluding the point $r = 0$ from the computational grid. However, one should be aware that these coordinates are no good anyway, since the coordinate lines in the original Cartesian system converge at the center point resulting in a very small step size $\Delta\theta$. We shall discuss this further in Sect. 17.4, where the same type of problem occurs at the poles of the globe when doing atmospheric simulations.

There are classic coordinate systems for many different types of geometry, and these should of course be used for computing. We saw above that the differential equation got a different and more complicated form. In Appendix A.2, the most

Fig. 10.14 Part of irregular domain and computational domain



common differential operators are listed in polar, cylindrical and spherical coordinates.

It may be possible to use structured grids even for other domains where no obvious coordinate system is available. In 2D, a general transformation is

$$\xi = \xi(x, y), \quad \eta = \eta(x, y),$$

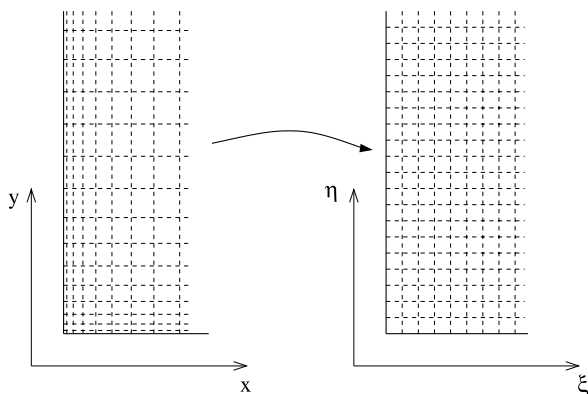
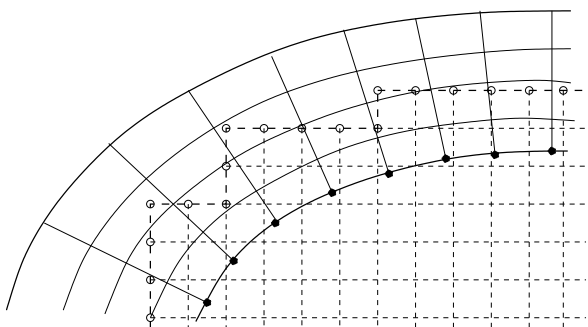
with a unique inverse transformation

$$x = x(\xi, \eta), \quad y = y(\xi, \eta).$$

The transformed differential equation contains derivatives of the variables x, y with respect to ξ, η and, if the transformation has been constructed by some numerical procedure, these are not known by any explicit expression. However, they can be approximated by finite differences, and in this way we can still keep a structured grid. If the final computation can be carried out on a uniform rectangular grid, we are in good shape, since the algebraic operations can be organized in an efficient way. This usually outweighs the extra complication caused by the fact that the differential equation contains more terms. There is also the advantage that the boundary can be represented exactly, with the exact form of boundary conditions. Figure 10.14 shows part of a domain with a curved boundary, and part of the computational domain.

In many applications, the solution varies on a much smaller scale in some parts of the domain than in others, which means that an efficient method should use a finer grid in those parts. A typical application where this occurs is fluid dynamics, where the solution may have very sharp gradients near solid walls. These are called *boundary layers*, and they require a fine grid. Figure 10.15 shows such a case with a simple geometry with the original coordinates to the left, and the computational coordinates to the right.

There is actually another way of handling irregular domains with curved boundaries. The problem is to construct a grid that is structured all over the domain. This difficulty is partially avoided by constructing a local grid near the boundary, and then couple it to one or more rectangular grids in the remaining part of the domain without requiring that the grid points match each other at the edges. This is called *overlapping grids*, and an example is shown in Fig. 10.16.

Fig. 10.15 Boundary layer grid**Fig. 10.16** Overlapping grids

The grid nearest the boundary is called a *curvilinear grid*, and is constructed such that the boundary conditions are easy to approximate. But there is now a new problem. Each one of computational domains has a new boundary, and since they are located in the inner part of the domain, there are no boundary conditions to pick up from the original problem. This is of course as it should be, since each grid must be coupled to the other one, and that coupling is obtained through new boundary conditions at these artificial boundaries. The most straightforward method is to define the boundary values by interpolation from the other grid. A certain point at the inner edge of the curvilinear grid (filled circle) is given a value that is interpolated from the nearest surrounding points in the rectangular grid. In the same way, a point at the outer edge of the rectangular grid (open circle) is interpolated from the nearest surrounding points in the curvilinear grid. The number of points used for interpolation is determined by the accuracy of the main difference scheme. A higher order scheme requires higher order interpolation using more points, otherwise the accuracy goes down.

Exercise 10.10 Write down the explicit difference scheme corresponding to (10.14) but now for the two-dimensional equation (10.16). Derive the stability condition.



<http://www.springer.com/978-3-642-19494-8>

Fundamentals of Scientific Computing

Gustafsson, B.

2011, XIV, 326 p., Hardcover

ISBN: 978-3-642-19494-8