CHAPTER 5

# Numerical Methods for Differential Equations

In this chapter we will discuss a few of the many numerical methods which can be used to solve initial value problems and one-dimensional boundary value problems. Today, users would only rarely develop or program algorithms themselves. Nonetheless, one must know what the algorithms and properties of various groups of methods are.

There are two groups of methods, namely single-step or multi-step methods. These are again divided into the sub-groups of explicit or implicit methods. The simplest of these will be mentioned here. One of their most important properties, their stability, will be investigated in an exemplary manner. The same will be done with the problem of numerical accuracy.

The discussion and the examples given are usually limited to a single differential equation of the general form we have derived before in Chapter 3:

$$\frac{dY(x)}{dx} = F(x,Y)\,, \quad Y(x_o) = Y_o \tag{1}$$

The independent variable here is *x*, which is usually equivalent to time in dynamical systems. *Y* represents the analytical solution of the differential equation, while *y* stands for the numerical approximation. The information gained through investigation of simple methods and examples can be used for other cases.

After revisiting some practical problems of numerical methods by playing with models introduced in Chapter 2, we will investigate the two most important properties of numerical algorithms—stability and accuracy. This is done most easily in the context of the simplest numerical method, i.e., the explicit Euler method (Section 5.2). The problem of solving stiff differential equations is addressed in the following section (Section 5.3), again by using Euler's method. Here we learn about the utility of implicit methods. Higher order methods are introduced in Section 5.4, and algorithms for step-size control are briefly mentioned in Section 5.5. Finally, in Section 5.6, we shall briefly turn to numerical methods for (one-dimensional) spatial systems, i.e., finite element methods.

An important point about numerical approximations of models of dynamical systems is often forgotten: They discretize our continuous differential equations. A numerical method applied to a continuous differential equation is a new model in its own right. You can demonstrate this for the logistic model of Section 2.2 (see also the CBT unit, Section 2.1.4; increase the step size of the numerical approximation and see what happens). In some cases the discrete model has nothing to do with the original system: it can show solutions which are not possible in the continuous model. Therefore, one of the important questions of numerical approximation theory is to what extent numerical models are similar to their original system of equations.

## 5.1 PRACTICAL PROBLEMS WITH NUMERICAL SOLUTIONS

In Chapter 2 we already encountered examples of dynamical models exhibiting numerical difficulties. We shall revisit three of these before we turn to a discussion of numerical methods used to deal with simple and complex cases. Here, the models will be presented using Simulink as the tool for modeling and simulation.

### 5.1.1 Three Communicating Tanks

The model of three communicating tanks grew out of the desire to model two tanks discharging into pipes converging at a node (Fig. 1). It turns out that this system cannot be modeled directly with most tools for dynamical systems: it leads to a closed loop which cannot be solved by most algorithms implemented in the modeling programs.

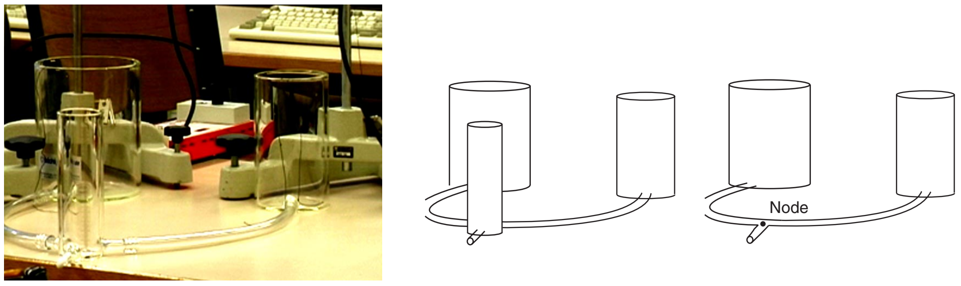

**Figure 1:** Two containers are connected by hoses to a third tank having a smaller cross section. From there the fluid drains through a short third pipe. The smaller tank replaces the node made up of two hoses converging into a single one.

Therefore, we decided to model the node as a true tank (Fig. 2). This results in a system which can easily be represented in simple tools such as Stella or Madonna. The numerical problem arises when we make the third tank increasingly smaller to represent a node in a system of pipes. It turns out that to simulate the equations with the help of fixed step numerical methods requires a time step so small that the solution procedure may become impractical.

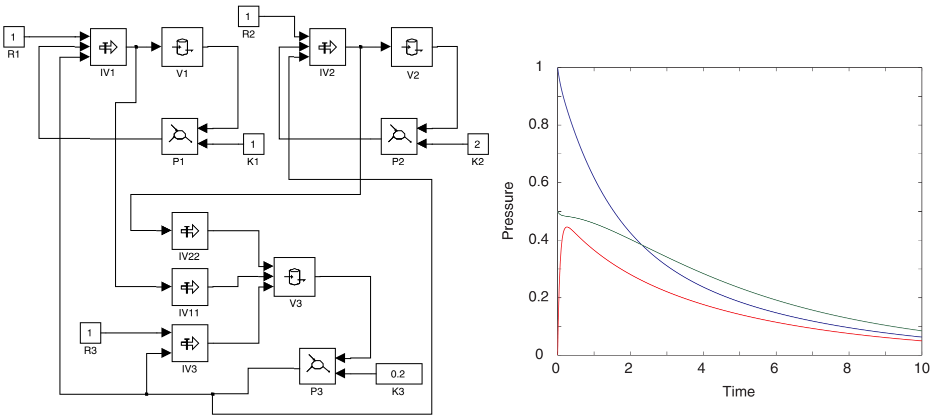

**Figure 2:** Simulink diagram of model of three communicating tanks. V1…V3 represent the stocks, while the quantities denoted by IV are the fluxes with respect to each of the tanks. In the simulation the cross section of the third tank is 5 to 10 times smaller than that of the other tanks. Units are arbitrary.

Assume that the first two tanks are filled while the third—representing the node—is empty at the beginning. If the capacitance of this last tank is made smaller and smaller, we obtain a model representing two tanks with the hoses joined at a node. In the model, the level of the fluid in the third tank rises very rapidly (Fig. 2). This gives rise to a model having time constants of strongly differing size: two large ones and one very small one.

Standard numerical methods—such as the ones built into Stella— have to use constant time steps smaller than the time in which the level in the “node” rises, which may be very small. This leads to an unacceptably large number of time steps over the entire interval of interest, and therefore to unacceptably long integration times. It is most interesting to see that the simple idea of integrating with a small time step for the short duration of the first rise, stopping the simulation and resuming it with new initial values and a new and larger time step, does not yield results. The simulation becomes just as unstable as if we had started the computation with a large time step to begin with. The information regarding the fast change is “wired” into the differential equations. Standard methods therefore fail.

A simulation carried out with one of the so-called "stiff" solvers in Simulink (or in Madonna) will yield good results very quickly. We shall learn in Section 5.3 what stiff differential equations and solvers are all about.

### 5.1.2 Two Blocks and Two Springs

You can build a model of two blocks oscillating on a horizontal surface. The first block is attached to a wall by a spring, and a second spring connects the first block to the second body. Here a result is presented in the form of a Simulink model (Fig. 3). The second block experiences friction, the other is driven by a harmonic force. If both springs are soft, the system is well behaved. There are two oscillatory frequencies of similar magnitude, making it easy for a simple solver such as the standard Runge-Kutta method of fourth order to give us acceptably accurate solutions with reasonable time steps.

If the spring connecting the two blocks is made very stiff, the behavior of the system changes. The second block should move with the first one as if it were connected by a rigid rod. On the other hand, the stiff spring leads to fast oscillations. As a result the second block performs fast damped oscillations around an average distance to the larger first block before it begins to move in tandem with the other body (Fig. 3). In the end, it is as if we had a single body connected to a single soft spring oscillating back and forth with a low frequency. The frequency is calculated from the total mass of the bodies and the spring constant of the remaining spring.

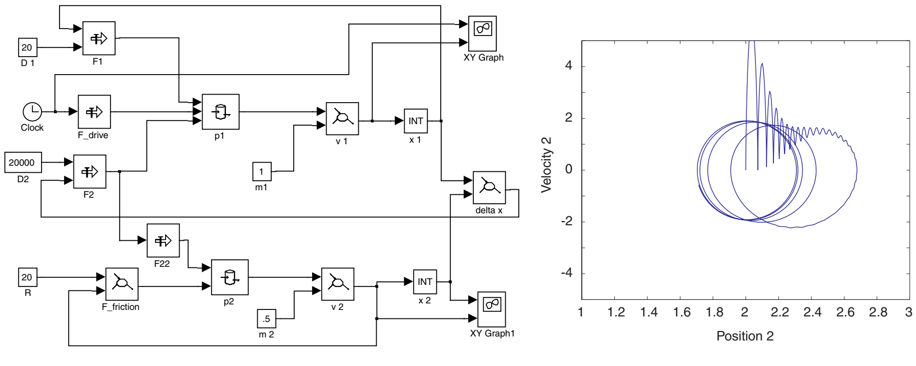

**Figure 3:** Simulink diagram of model of two blocks and two springs. Note the stocks for the momenta of the bodies, and the integrators for the positions. The results of the simulation are shown on the right as a phase plot. The spring constants are 20 and 20,000, respectively. Units are arbitrary. If the second spring is made very stiff, the numerical solution becomes difficult to compute for standard methods.

As in the case of the three communicating fluid tanks, the model becomes stiff if the value of the second spring constant is increased. Now we understand the term “stiff.” It originates in mechanical systems where connecting springs are made stiff to represent rigid rods. Standard numerical methods cannot properly deal with such systems, and trying to circumvent the problem by restarting the integration with a larger time step after the fast changes have died down does not work. Only solvers suited for stiff models are practical in this case.

### 5.1.3 Conduction of Heat in a Long Bar

Let us have a brief look at an entirely different type of model. In Section 2.3 we created the model representing the flow of heat through a long conducting bar. There you can observe a possibly surprising feature of instability. For a given number of elements into which the bar has been divided, the solution using Euler’s method becomes unstable for a certain time step (Fig. 4). We might expect that for double the number of elements, i.e., for a finer spatial approximation, we would be rewarded with a larger time step for which the simulation remains stable. The opposite is true. Instead of a larger time step, we need a smaller one. In fact, numerical experiments show that for twice the number of spatial elements we have to reduce the time step by a factor of four.

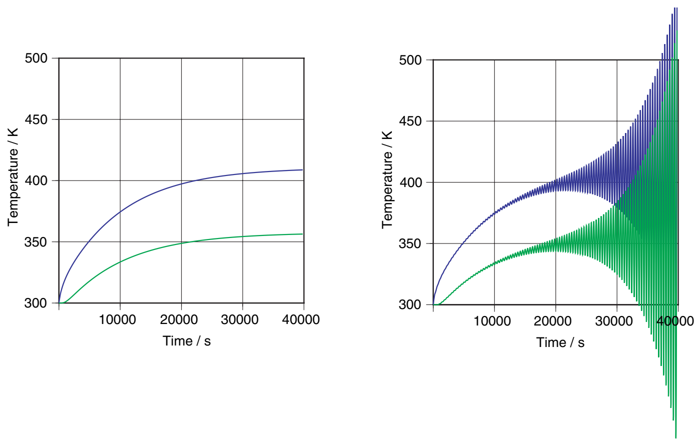

**Figure 4:** Solution of “finite element” model having five elements in a copper bar of 1 m length. Both solutions have been computed with Euler’s method. In the first, a time step of 100 s has been used, in the second the time step was 200 s. We see the typical appearance of unstable solutions.

From a physical viewpoint the result is not that surprising. Having elements only half as long makes their thermal inertia smaller by a factor of two. In addition, the thermal resistance will be halved as well. Together these factors lead to time constants in the model which are reduced by a factor of four.

In summary, solving model equations numerically can pose moderate to grave problems. In simple cases, we check the accuracy of solutions by decreasing the time step and making sure that the solutions do not vary any longer—at least not noticeably. In difficult cases such as stiff systems of equations we have to make sure we use appropriate algorithms which include strong stability and automatic step-size control. All in all, numerical analysis of dynamical (and spatial) systems is a practical art.

## 5.2 STABILITY AND ACCURACY OF NUMERICAL SOLUTIONS

Stability and accuracy are two of the most important practical features of numerical methods for differential equations. The problems posed by these features can be investigated by using just a single numerical method—Euler's method. It is simple and allows for simple theoretical investigations of the phenomena of approximation of initial value problems. Here we will present the method, discuss its stability properties and its error behavior. The latter leads to the idea of extrapolation which can be used as an effective step-size control algorithm.

In the following we are going to use Euler's method mostly on the single linear homogenous initial value problem

$$\frac{dY}{dt} = \lambda Y \quad , \quad Y(0) = 1 \tag{2}$$

Here, $\lambda$ is a real valued number. If $\lambda < 0$, the differential equation Eq. (2) is said to be stable, for $\lambda > 0$ it is unstable. (It is important to distinguish the stability of the analytical solution of an initial value problem from the stability properties of a numerical approximation.)

### 5.2.1 The Explicit Euler Method

The simplest example of a numerical method for initial value problems is the explicit Euler method. The differential quotient in Eq. (1) is replaced by the difference quotient (where $\Delta x = h$ is used) and the function $F$ by its value at the beginning of the (time) step:

$$y_{i+1} = y_i + hF(x_i, y_i) \tag{3}$$

In just one step a new numerical value $y_{i+1}$ is calculated at $x_{i+1} = x_i + h$. It is obtained from the previously calculated value $y_i$ at $x_i$ (see Fig. 5).

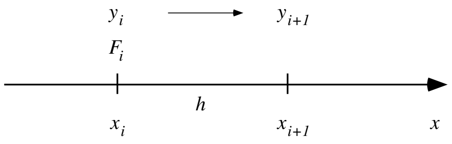

**Figure 5:** Representation of the axis of the independent variable ($x$). In a single step method, an approximate solution is calculated at a point on the basis of the already known solution at the previous point. All quantities necessary for computation are known at $x_{\mathrm{i}}$; $h$ is the step length.

Because the function $F$ is taken at the previous position, it is known explicitly. Even when the differential equation is non-linear (when $F(x,Y)$ is a non-linear function) the Euler approximation is linear and explicit in $y_{i+1}$ and therefore easy to solve. Graphically, the Euler method corresponds to a continuous creation of tangents to the analytic solution curve with an initial value of $y_i$ (see Fig. 6; the true and the Euler solution for Eq. (2) with $\lambda = -1$ and $h = 0.5$ are shown there). $F(x_i,y_i)$ is equivalent to the tangent to the curve at the corresponding point. It is clearly visible how the numerical solution deviates from the true solution. One can imagine that this deviation must be smaller if the step $h$ were made smaller. The behavior of the approximation error will be investigated more closely in Section 5.2.3.

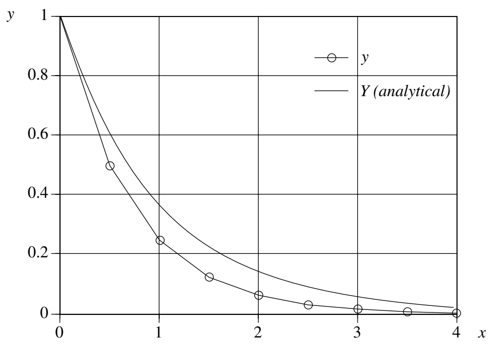

**Figure 6:** Comparison of the Euler solution of the initial value problem (2) with the analytical solution.

### 5.2.2 Stability of the Euler Method

Two essential features of numerical methods for initial value problems must be distinguished from each other. These are the error and stability properties. The second point will be dealt with here in an exemplary manner by comparing the explicit and implicit Euler methods (the implicit method will be discussed below in Section 5.3.2). The knowledge gained can, in many cases, be transferred directly to higher order methods.

Fig. 7 demonstrates the stability problem of the explicit Euler method for the simple example of Eq. (2). Eq. (2) with $\lambda = -1$ was solved using a step length of $h = 2.5$. The numerical solution has absolutely nothing to do with the true solution which is shown in the diagram as well. It oscillates and increases exponentially, i.e., it "explodes." One says that it has become unstable. If the example is solved using various step lengths, one sees that the method oscillates if $h > 1.0$, and becomes unstable when $h > 2.0$.

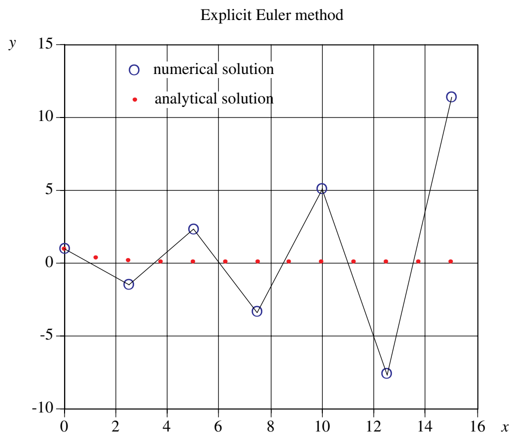

**Figure 7:** Numerically unstable solution of the differential equation (9) using the Euler method. Instability results from steps which are too large and as a result of the features of the explicit Euler method.

We can calculate the numerical solution of the test example in Eq. (2) formally. If Eq. (2) is solved with Euler's method, $y_{i+1}$ is obtained from $y_i$ as follows:

$$y_{i+1} = y_i(1 + \lambda h) \tag{4}$$

Starting with an initial value $y_o$, the numerical value $y_N$ after $N$ steps is

$$y_N = y_o(1 + \lambda h)^N \tag{5}$$

We see that the numerical values increase for $\lambda = -1$ if $h > 2$. In this case the absolute value of the factor $|1+\lambda h|$ is larger than 1. Remember that for negative values of $\lambda$ the analytical solution decreases exponentially.

### 5.2.3 Accuracy of Numerical Approximations with Euler's Method

Numerical error behavior, i.e. the properties of the numerical approximation, of a method is, along with stability, one of its most important aspects.The approximation error should be kept as small as possible while, at the same time, the effort calculating a solution should not be unacceptably large. Methods of different orders and different types have different error behavior (see Section 5.4). Their efficiency, meaning the number of calculations needed for a given accuracy, is usually closely connected to the order. Nevertheless, methods of the same order can exhibit rather different properties.

Knowledge of the error properties is especially important for controlling the step-lengths during the integration (see Section 5.2.5 and Section 5.5). The efficiency of the step-size control mechanism of a method is decisive for the number of calculations which are needed for simulating a model.

As usual, we choose the simplest method to learn about the behavior of numerical methods. In investigating the error behavior of methods for initial value problems the question arises of how the approximation error over a fixed integration interval depends upon the length $h$ of the steps used. The error per step is called *local error*, whereas the error at the end of a fixed interval is called *global error*. We use these informal definitions:

> *The local error is the result of a computation of a single step starting with an accurate initial value. The global error, on the other hand, is the result of the propagation of the errors occurring at every step.*

Table 1 shows the local error of the Euler method for $dY/dx = -Y$ for just one step of length $h$ with a starting value $Y_0 = 1$ at $x = 0$. We see that the difference between the true and the numerical solutions is smaller by a factor of 4 when the integration step is halved. This indicates that the local approximation error of the Euler method reduces quadratically with $h$. A little theory confirms this experimental result. A true value of the solution $Y$ of the differential equation $dY/dx = F(x,Y)$ at $x = h$ is obtained when $F(x,Y)$ is integrated from $x = 0$ to $x = h$:

$$Y(h) = Y_o + \int_0^h F(x,Y)dx \tag{6}$$

The function $F(x,Y)$ can be expanded at $x = 0$ in a Taylor's series. If the result is inserted into the previous relationship Eq. (6), we obtain:

$$Y(h) = Y_o + \int_0^h \left( F(x_o) + x\left.\frac{dF}{dx}\right|_{x_o} + \frac{1}{2}x^2\left.\frac{d^2F}{dx^2}\right|_{x_o} + \dots \right) dx$$

$$= Y_o + hF(x_o) + \frac{1}{2}h^2\left.\frac{dF}{dx}\right|_{x_o} + \frac{1}{6}h^3\left.\frac{d^2F}{dx^2}\right|_{x_o} + \dots$$

If the numerical solution $y(h) = Y_0 + hF(x_0)$ is subtracted, we obtain:

$$Y(h) - y(h) = \frac{1}{2}h^2\left.\frac{dF}{dx}\right|_{x_o} + \frac{1}{6}h^3\left.\frac{d^2F}{dx^2}\right|_{x_o} + \dots \tag{7}$$

In Table 1 the error of the Euler method and the first term on the right hand side of Eq. (7) are given for various step-lengths. The local error not only reduces with the square of $h$, but, for our example, it agrees very well with the first term of the theoretical result in Eq. (7). This only works here because the higher order terms in Eq. (7), i.e. the higher order derivatives of $F$, are relatively small. Unfortunately, one cannot generally use the first term of this Taylor-series form of the approximation as a good error estimate.

| h | $Y(h) - y(h)$ | $h^2 dF/dx/2$ |
|---|---|---|
| 0.2 | 1.87E–02 | 2.00E–02 |
| 0.1 | 4.84E–03 | 5.00E–03 |
| 0.05 | 1.23E–03 | 1.25E–03 |
| 0.025 | 3.10E–04 | 3.13E–04 |
| 0.0125 | 7.78E–05 | 7.81E–05 |
| 0.00625 | 1.95E–05 | 1.95E–05 |
| 0.003125 | 4.88E–06 | 4.88E–06 |

**Table 1:** Local error of the approximate solution with Euler's method for $dY/dx = -Y$, $Y_0 = 1$.

Formally, one expresses the result obtained for the local error as follows. If $Y(h) - y(h)$ is the local error, we say that

$$Y(h) - y(h) = C_2 h^2 + O\left(h^3\right) \tag{8}$$

$C_2$ is a factor dependent upon the method, and we use $O(h^3)$ to denote the terms of order 3 and higher.

The global error, i.e., the error which results in several steps over a fixed interval, depends upon the local error (error per step) as well as the number of steps in the entire interval of integration $x_N - x_0$ (here $N$ is the number of steps of length $h = (x_N - x_0) / N$). We assume that the error per step for steps of the same length, is always about the same. Thus the global error is $N$ times as large as the local error. Because of this, the global error decreases with a power of $h$ which is $p - 1$, where $p$ is the order of the single-step error. In our case

$$Y(x_N) - y_N = C_1 h + O(h^2) \tag{9}$$

Again, $C_1$ is a factor dependent upon the method, and $O(h^2)$ denotes the terms of higher order. The results in Table 2 confirm this: if the integration step is halved, the error decreases by a factor of two.

| **Steps** | $y_N$ | $Y(1) - y(1)$ |
|---|---|---|
| 5 | 0.327680 | 0.040199 |
| 10 | 0.348678 | 0.019201 |
| 20 | 0.358486 | 0.009394 |
| 40 | 0.363232 | 0.004647 |
| 80 | 0.365568 | 0.002311 |
| 160 | 0.366727 | 0.001153 |

**Table 2:** Global error of Euler's method for the simple equation $dY/dx = -Y$ (with an initial value $Y_0 = 1$ at $x = 0$), at $x = 1$.

### 5.2.4 Extrapolation of Euler's Method

The observation that the global error of the Euler method decreases linearly with the step-length, allows for error estimates with a simultaneous correction of the numerical result (extrapolation). If the numerical result is calculated twice from the starting point until the end of the interval, first with $N$ steps and again with twice the number of steps ($2N$), the error in the second result will be about half of what it is in the first. The difference between the two numerical values is an approximation of the error. If the leading term $Ch$ is taken as the error according to Eq. (9), the results of the two solutions are

$$Y - y_N = C\frac{x_{final} - x_o}{N}$$

$$Y - y_{2N} = C\frac{x_{final} - x_o}{2N}$$

If $C$ and the interval of integration are eliminated, we get the error of the better solution (the one calculated with $2N$ steps):

$$Y - y_{2N} \approx y_{2N} - y_N \tag{10}$$

This simply means that the error of the numerical method, which is calculated with $2N$ steps, is estimated by the difference of the two approximate values $y_{2N}$ and $y_N$. Because we now have an error estimate, we can correct the better solution and obtain an even better numerical approximation. In Table 3, the numerical solutions of our test example for step numbers increasing by a factor of two are calculated. Column 3 shows the true error (remember that the true solution in Column 1 is known). Column 4 shows the differences of consecutive solutions which represent an estimate of the approximation error of Euler method. This error is used to calculate an extrapolated value (Column 5). The extrapolated value is indeed much better than the one obtained by Euler's method.

| **Steps** | $y_N$ | $Y(1) - y_N$ | $y_{2N} - y_N$ | $y$**(extrap.)** |
|---|---|---|---|---|
| 5 | 0.327680 | 0.040199 | | |
| 10 | 0.348678 | 0.019201 | 0.020998 | 0.369676 |
| 20 | 0.358486 | 0.009394 | 0.009808 | 0.368294 |
| 40 | 0.363232 | 0.004647 | 0.004746 | 0.367978 |
| 80 | 0.365568 | 0.002311 | 0.002336 | 0.367904 |
| 160 | 0.366727 | 0.001153 | 0.001159 | 0.367886 |

**Table 3:** Error estimate and first extrapolation at $x = 1$ of the solution of $dY/dx = -Y$, $Y_0 = 1$, using Euler's method.

If one observes the form of the error of Euler's method, i.e., Eq. (9), one sees that the leading error term is missing in the series of extrapolated solutions in Column 5 of Table 3. The values of this series should converge more quickly. That means that it converges with a higher power of the step length $h$. Indeed, Column 4 in Table 4 corresponds to a solution of a second order method. In such a method, the error resulting from decreasing the step by a factor of two, is smaller by a factor of 4. The method of extrapolation described above can now be used for this column of numerical values. The difference between two consecutive values is approximately three times the approximation error of the more exact solution. Using this estimate of the error, a further extrapolation step can be carried out. This is done in Column 6 of Table 4. Theoretically, this method can be continued indefinitely but it becomes impractical because of round off errors.

Error estimation tends to be a difficult subject. In particular, trying to determine global errors is not easy. The method with which an initial value problem is calculated twice using different step lengths, and to compare the results, is arguably one of the simplest and fastest. In combination with extrapolation, it is even more attractive. This does not mean, though, that it is the only possibility for error estimation (see Section 5.5)

| Steps | $y_N$ | $y_{2N} - y$ | $y^*$(extrap.) | $y^*_{2N} - y^*$ | $y$(extrap.) |
|---|---|---|---|---|---|
| 5 | 0.327680 | | | | |
| 10 | 0.348678 | 0.020998 | 0.369676 | | |
| 20 | 0.358486 | 0.009808 | 0.368294 | – 0.001382 | 0.36783333 |
| 40 | 0.363232 | 0.004746 | 0.367978 | – 0.000316 | 0.36787267 |
| 80 | 0.365568 | 0.002336 | 0.367904 | – 0.000074 | 0.36787933 |

**Table 4:** Continued extrapolation of the solution with Euler's method of $dY/dx = -Y$ at $x = 1$ (with initial value $Y_0 = 1$ at $x = 0$).

### 5.2.5 Automatic Step-Size Control

Automatic step-size control can be very important for making numerical algorithms practical. Even in non-stiff cases where the standard methods built into most modeling tools suffice, the numerical solution can become impractical to carry out if the smallest time step necessary in the integration has to be used for the entire interval.

Naturally, automatic time-step control necessitates automatic error estimates. The size of a time step has to be small enough to ensure that the local error made in this step is below a chosen tolerance.

Having used extrapolation on Euler's method, we can devise a simple algorithm for error estimation and step-size control. Starting at a point $x_i$, a single step of length $h$ is carried out to $x_{i+1}$, yielding a result $y_1$. Then a second solution at $x_{i+1}$ is calculated using two steps of length $h/2$ resulting in an approximation $y_2$. The error of the second solution is now estimated according to Eq. (10):

$$e_{abs} = |y_2 - y_1| \tag{11}$$

$e_{abs}$ is the *absolute error* of $y_2$. If we want to base the step size control on the *relative error* instead, we have to calculate

$$e_{rel} = \frac{|y_2 - y_1|}{y_2} \tag{12}$$

Now the estimate of the error is compared to an absolute or relative tolerance *tol*. Since Euler's method is a method of first order (the local error decreases quadratically with $h$), we can calculate an optimal step-size by

$$h_{opt} = h\left(\frac{tol}{e}\right)^{1/2} \tag{13}$$

If the results calculated so far are larger than the tolerance, we discard them and recalculate $y_1$ and $y_2$ with $h_{opt}$. Otherwise we accept them and use $h_{opt}$ as our new step size. Naturally we could also extrapolate the solution and get

$$\hat{y}_2 = 2y_2 - y_1 \tag{14}$$

As we have seen before, this value is much more accurate than $y_2$. An estimate of the local error of this solution could be obtained as well.

What we have discussed here is a strongly simplified version of a step-size control algorithm. In general, much more care has to be exercised, mixed criteria have to be used, and safety factors have to be built into the algorithm. We will discuss these points below in Section 5.5.

## 5.3 STIFF DIFFERENTIAL EQUATIONS

Stiff differential equations pose some of the biggest challenges to numerical methods for initial value problems. As we have seen in our examples, standard methods such as the explicit Euler Method and its higher order cousins are inherently unstable. Codes trying to adjust the time step to larger values in smooth regions of the solutions become immediately unstable and fail. Stiff problems therefore call for different numerical methods. As we shall see, a simple change to Euler's method—namely to an implicit version—yields an algorithm which is very stable even for the most difficult dynamical systems. Higher order versions of such implicit codes can be very stable and efficient.

### 5.3.1 What Are Stiff Differential Equations?

There is no unique or simple answer to the question of what stiffness is all about. We have seen that a feature which comes up again and again—but not necessarily all the time—is the occurrence of fast and slow components in the same system. Examples are chemical reactions where some steps run very fast while others are slow. In oscillatory cases, stiffness may arise from the mixing of fast and slow oscillations.

Yet these observations are not general enough to make for a good definition. Stiffness may depend on the systems—the equations—themselves, or on the particular initial values, or on the interval of integration.

Therefore, it is best to stay with a simple and practical "definition" of stiff differential equations. Systems are stiff if explicit numerical methods fail. Put differently, stiff problems are those for which implicit methods work very well and efficiently.

### 5.3.2 The Implicit Euler Method

The Euler method discussed until now was the explicit method. The unknown numerical quantity $y_{i+1}$, i.e., the components of the corresponding vector, can be calculated by an explicit equation. This means that the equations of a system are all individually and directly solvable using the components $y_{i+1}$ at the new point $x_{i+1}$.

There are numerical methods, though, which do not allow for such direct solutions. The simplest example is, again, the Euler method. This time it is the *implicit* method:

$$y_{i+1} = y_i + hF(x_{i+1}, y_{i+1}) \tag{15}$$

The only important difference to the explicit Euler method is this. The right hand side of the system of differential equations (Eq. (1)) is computed at the new point $x_{i+1}$ (see Fig. 8). A single linear differential equation such as Eq. (2) can still be solved explicitly. However, systems of linear equations and nonlinear equations lead to implicit systems of algebraic equations. A system of nonlinear IVPs necessarily leads to a system of nonlinear algebraic equations for the components $y_{i+1}$. We normally use the Newton-Raphson method for solving systems of nonlinear algebraic equations. The numerical effort necessary for solving such systems of equations is much larger than for explicit methods. Therefore, introducing implicit methods appears to be impractical. We shall see, however, that implicit methods have desirable stability characteristics for some very difficult problems which are lacking in explicit algorithms. This may make the method efficient even though the solution algorithms are more complex.

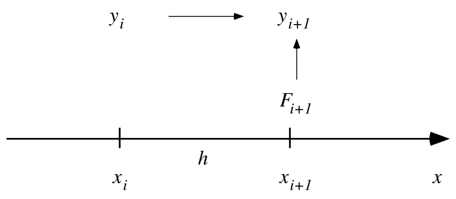

**Figure 8:** Representation of the axis of the independent variable for the implicit Euler method.

## 5.3.3 The Stability of the Implicit Euler Method

The implicit Euler method is very different from the explicit one in its stability properties. In solving the same example Eq. (2), one sees that the method stays stable for any step length, even large ones (Fig. 9). Note, however, that the accuracy of the solution may be bad if we choose a large step $h$. This simple example shows that stable solutions may not have to be accurate, but accurate solutions will be stable. Therefore we always have to satisfy stability before accuracy.

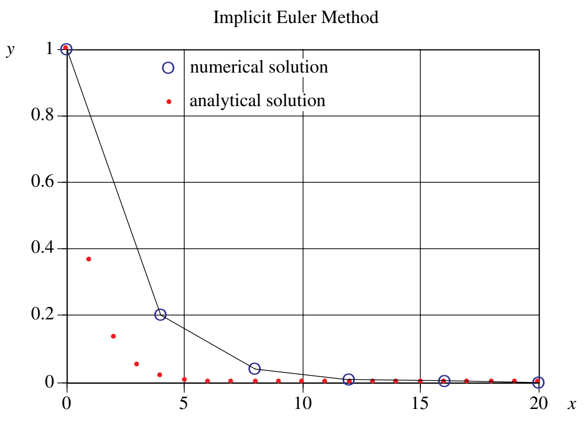

**Figure 9:** The implicit Euler method gives a stable solution for steps which even may be too large for an accurate solution. The size of the step only effects the accuracy of the result.

Again we can calculate the numerical solution of Eq. (2) by hand. In the case of the implicit Euler method, the corresponding solution is:

$$y_{i+1}(1-\lambda h) = y_i \tag{16}$$

After $N$ steps starting with an initial value $y_o$ we have

$$y_N = \left(\frac{1}{1-\lambda h}\right)^N y_o \tag{17}$$

For negative values of $\lambda$, the numerical solution $y$ approaches zero independent of the value of $h$, as should be expected of the exponentially decreasing analytical solution.

The stability behavior which we have witnessed in the case of the explicit and implicit Euler methods can more or less be transferred to methods of higher order. Explicit methods always have a limited stable range for the integration step $h$. Outside of this range, every numerical solution is unstable. This is true for single-step as well as multi-step methods (including predictor-corrector-methods, see Section 5.4). Collocation methods and their implicit Runge-Kutta counterparts are strongly stable for any value of $h$.

Combined with an efficient step-size control mechanism which allows the step size to be adjusted according to accuracy requirements (rather than being limited by stability), the stability of implicit methods makes them strong candidates for systems of stiff differential equations.

## 5.4 SOME EXAMPLES OF NUMERICAL METHODS

So far we have discussed the most relevant features of numerical methods for Euler's scheme only. Now we will turn to some of the many numerical algorithms known today. Among these are explicit and implicit Runge-Kutta methods (collocation methods) which are single-step algorithms, and so-called multi-step methods. Again, stability and accuracy are the main properties we are interested in. As with Euler's method, explicit codes have strongly limited stability features, while implicit schemes are inherently stable. Later in Section 5.5 we shall discuss error estimation and step-size control for some of the examples.

### 5.4.1 Runge Kutta Methods

A simple second order method can be constructed as follows. The new value $y_{i+1}$ is calculated with the help of the function values $F_i$ and $F^*_{i+1}$. The value of the function $F$ at $x_{i+1}$ is obtained with the help of the value of $y^*_{i+1}$ which is calculated first by using an Euler step:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{2}h(F(x_i, y_i) + F(x_{i+1}, y*_{i+1})) \\ y*_{i+1} &= y_i + h \cdot F(x_i, y_i) \end{aligned} \tag{18}$$

This algorithm is called the Heun method. It is an explicit single step method, just like the explicit Euler method, but more accurate. While the Euler method is a first order method, Heun's algorithm is of the second order (sometimes called two-stage; the global error is of second order as we shall see in Section 5.4.4). Thus, more calculations per step are needed. In practice, the question arises of the relationship between accuracy and the amount of computational work, i.e., the efficiency of numerical methods (Section 5.4.5).

Both of the methods described here (Euler and Heun) are examples of a large class called explicit Runge-Kutta methods (ERK). There are numerous ERK methods of various orders. They are certainly the most popular single step methods. Good and efficient ERKs have been designed lately.

A further method, the four-stage "Classic RK Method" of fourth order, found in almost every program for dynamical systems, is still to be mentioned here:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6}h\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ k_1 &= F\left(x_i, y_i\right) \\ k_2 &= F\left(x_i + h/2, y_i + h/2k_1\right) \\ k_3 &= F\left(x_i + h/2, y_i + h/2k_2\right) \\ k_4 &= F\left(x_i + h, y_i + hk_3\right) \end{aligned} \tag{19}$$

A general way of representing the Runge-Kutta methods has been established (Hairer, Norsett, Wanner, *Solving Ordinary Differential Equations I*, p.132-133):

$$\begin{aligned} y_{i+1} &= y_i + h\left(b_1k_1 + \ldots + b_sk_s\right) \\ k_1 &= F\left(x_i, y_i\right) \\ k_2 &= F\left(x_i + c_2h, y_i + ha_{21}k_1\right) \\ k_3 &= F\left(x_i + c_3h, y_i + h\left(a_{31}k_1 + a_{32}k_2\right)\right) \\ &\;\,. \\ &\;\,. \\ k_s &= F\left(x_i + c_sh, y_i + h\left(a_{s1}k_1 + \ldots + a_{s,s-1}k_{s-1}\right)\right) \end{aligned} \tag{20}$$

In general, the methods are called $s$-stage methods. The coefficients which appear in the method are shown in a pattern called the matrix of coefficients (Table 5).

| | | | | | |
|---|---|---|---|---|---|
| 0 | | | | | |
| $c_2$ | $a_{21}$ | | | | |
| $c_3$ | $a_{31}$ | $a_{32}$ | | | |
| … | … | … | | | |
| $c_s$ | $a_{s1}$ | $a_{s2}$ | … | $a_{s,s-1}$ | |
| | $b_1$ | $b_2$ | … | $b_{s-1}$ | $b_s$ |

**Table 5:** Matrix of coefficients of explicit Runge-Kutta (ERK) methods.

The three single step methods mentioned here—Euler's method, Heun and the Classic Runge Kutta algorithm—are neither the only nor the best, but they are relatively simple, very well known and often used (these are the three methods which are found in the Stella program). In more advanced codes, however, other methods are used which include error estimation and step-length control. The differences between the methods, especially their error orders and stability properties, will become clearer in Section 5.4.4.

### 5.4.2 Collocation Methods (Implicit Runge-Kutta Methods)

An important class of implicit single-step methods are the collocation methods. The implicit Euler method is the simplest method in this class. Because they are of very general form, they are interesting for both practice and theory (they have become important in finite element methods as well). They possess excellent stability properties and exhibit what is known as super convergence, making them very accurate (Section 5.4.4).

In collocation methods, as in general finite element methods, the solution is approximated by a simple function, often a polynomial of a certain order $n$:

$$y(x) = \sum_{j=0}^{n} d_j \Phi_j(x) \tag{21}$$

The unknown coefficients are determined as follows. The approximation (21) is introduced into the differential equations and the residuals (the error incurred because of the approximation) is set equal to zero at $n$ selected points in the interval of integration. In other words, we satisfy the differential equations with the approximation introduced at $n$ distinct points—the *collocation points*—in the interval $[x_i,x_{i+1}]$:

$$\left.\frac{dy}{dx}\right|_{x=x_k} = F(x_k, y_k) \quad , \quad k = 1 \ldots n \tag{22}$$

Different collocation methods are distinguished by different choices of the collocation points (Fig. 10). If $x_{i+1}$ is included but $x_i$ is not, we obtain the most stable group in the class of collocation methods (the implicit Euler method is an example in this group).

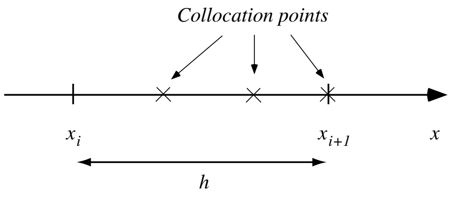

**Figure 10:** In the collocation method, the residual resulting from introducing the approximation (21) in the differential equation is set equal to zero at selected points in the interval of length $h$. The end points $x_i$ and $x_{i+1}$ may be collocation points. If the so-called Radau-Points are used, $x_{i+1}$ is included, but $x_i$ is not.

There are $n$ collocation points $x_k$, $k = 1 \ldots n$, in the integration interval $[x_i,x_{i+1}]$ (see Fig. 10). To determine the $n$+1 coefficient $d_j$ in Eq. (21), the initial value $y_i$ and $n$ collocation equations (22) are needed. Eq. (21) allows interpolation of the desired solution in an interval after the coefficients have been determined.

The choice of collocation points is decisive for the accuracy and stability of the method. Investigations have shown that the Gauss-, Lobatto-, and Radau points known from numerical integration, are the most appropriate because the approximation error is of especially high order (superconvergence, Section 5.4.4). For the problem of stiff differential equations, the Radau collocation method has proven to be the most useful because of its strong stability properties (the implicit Euler method is the Radau collocation method of the first order; in Fig. 10 the collocation points for the Radau method of the third order are shown). As with all implicit methods, systems of nonlinear algebraic equations can result here. The large effort required for solving these is considered worthwhile only when the methods have other important advantages, such as strong stability.

To break the entire interval of integration down into small elements and approximate the solution by simple functions in each element, is called a finite element method (Section 5.6).

### 5.4.3 Multi-step Methods

In so-called multi-step methods, a numerical approximation of a new value $y_{i+1}$ at $x_{i+1}$ can be calculated by a combination of values of $y$ and $F$ at several previous points. A simple example for such a method is the explicit mid-point rule

$$y_{i+1} = y_{i-1} + 2hF(x_i, y_i) \tag{23}$$

Here the value $y_{i+1}$ is dependent upon the previously calculated numerical values $y_{i-1}$ and $y_i$ at the two points $x_{i-1}$ and $x_i$ (see Fig. 11).

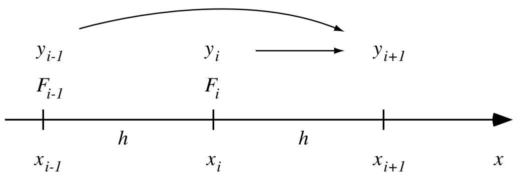

**Figure 11:** Shown here is the axis of the independent variables in a multi-step method (here a two-step method is shown). The approximate solution at point $x_{i+1}$ is calculated on the basis of values at several previous points.

A further example of a third order multi-step method (which is called the explicit Adams method) is

$$y_{i+1} = y_i + h\left(\frac{23}{12}F_i - \frac{16}{12}F_{i-1} + \frac{5}{12}F_{i-2}\right) \tag{24}$$

Other combinations of values of $y$ and $F$ at previous points, lead to methods of different orders which have different accuracy and stability properties.

Just as with single-step methods, we can design implicit algorithms for multi-step methods as well. Here is just one example (an implicit Adams method):

$$y_{i+1} = y_i + h\left[\frac{5}{12}F_{i+1} + \frac{8}{12}F_i - \frac{1}{12}F_{i-1}\right] \tag{25}$$

Because of the large number of calculations needed for implicit methods, so called predictor corrector methods are often used. The value of $y_{i+1}$ is calculated first with the help of an explicit Adams method (such as Eq. (24)), whereupon in Eq. (25) $y_{i+1}$ can be explicitly calculated. Unfortunately, the strong stability properties, which are the reason implicit methods are so often used, can be lost in this process.

Multi-step methods usually possess high accuracy and may involve fewer computations than single-step methods (such as ERK) of an equivalent order. On the other hand, they are not self-starting. A multi-step code has to be combined with a single-step method to calculate the first few numerical values necessary for a multi-step algorithm. Also, if variable steps have been used to calculate the previous values, interpolation is needed to obtain values at fixed steps $h$ (note that the intervals between values used in a multi-step method have to be equidistant). This may make the codes more complex and less efficient.

### 5.4.4 Stability and Accuracy of Higher Order Methods

Higher order methods are developed with the intention of decreasing the approximation error. In practice, one attempts to find procedures for which the Taylor series development of the (local) error contains terms of higher order only. As a consequence, the error should decrease much faster than with a first order method (such as Euler's method) if the integration step is decreased. With a method of the 2nd order it would be expected that for example, when a step is halved, the local error would decrease by a factor of 8 (third order). The global error, on the other hand, would decrease by a factor of 4 (second order). In Table 6, the global approximation errors of the Euler method, the Heun procedure and the classical Runge-Kutta method are presented. It is clear to see that the errors of the Euler method are proportional to $h$, while those of the Heun method change quadratically with $h$, and those of the Runge-Kutta method change with the fourth power of the step size.

| Number of steps | Euler | Heun | CRK-4 |
|---|---|---|---|
| 5 | 4.02E–02 | –2.86E–03 | –5.80E–06 |
| 10 | 1.92E–02 | –6.62E–04 | –3.33E–07 |
| 20 | 9.39E–03 | –1.59E–04 | –2.00E–08 |
| 40 | 4.65E–03 | –3.90E–05 | –1.22E–09 |
| 80 | 2.31E–03 | –9.67E–05 | –7.56E–11 |

**Table 6:** Global error of the simplest three explicit single-step methods for $dY/dx = -Y$ at $x = 1$(with initial value $Y_0 = 1$ at $x = 0$).

Quite often, but not always, an $s$-stage method has a global error of order $s$, while the local error is of order $s$+1. There are exceptions, however. An example of an $s$-stage method having a global error of order 2$s$–1 (at the end points $x_{i+1}$, not in the intervals of length $h$, though) is the (implicit) Radau collocation method. Thus the 3-stage Radau method has a global error at the ends of the integration steps which decreases with the 5th power of $h$. This feature of accelerated convergence is called super-convergence.

Stability is again an altogether different issue. To put a complex story simply, explicit algorithms have limited stability domains (the step $h$ can be increased only up to a limited value), while Radau collocation methods—including the implicit Euler method—have the strongest stability properties. These latter methods are absolutely stable and strongly damp possible numerically induced oscillations.

Let us consider the stability of the simplest ERK methods more closely. This will demonstrate why the explicit Euler method behaves so badly for oscillatory problems where the classical 4-stage Runge-Kutta method is quite appropriate in many cases.

We now have to relax the requirement that $\lambda$ in our test example should be real. In a system of (linear) differential equations

$$\frac{d\mathbf{Y}}{dt} = \mathbf{AY} \tag{26}$$

showing oscillatory behavior, the eigenvalues of the matrix $\mathbf{A}$ are complex. We therefore consider a test equation (as in Eq. (2)) having complex valued $\lambda$. As before, for the explicit Euler method, we have Eq. (4):

$$y_{i+1} = y_i(1 + \lambda h)$$

This means that the absolute value of $1 + \lambda h$ must be smaller than 1, i.e.,

$$|1 + \lambda h| < 1 \tag{27}$$

This restricts $\lambda h$ to a circle of radius 1 centered at –1 in the complex plain (see Fig. 12). For an undamped oscillation of a body hanging from a linear spring, the eigenvalues of the matrix $\mathbf{A}$ are purely imaginary and so is $\lambda h$ (a negative real part appears if the oscillation is damped). Inspection of Fig. 12 shows that imaginary $\lambda h$ do not lie within the domain of stability (the circle mentioned before), independent of the value of $h$. This means that the Euler method cannot be stable for such problems.

The stability domains of the Heun and classical Runge-Kutta methods, on the other hand, include part of the imaginary axis in Fig. 12. Apart from the larger stability domain, we expect the classical ERK method to be all right on undamped oscillatory problems. This is what you can try out in our simple practical applications presented and discussed in Chapter 2 of the CBT materials. This means that we should turn to a higher order method for the numerical solution as soon as we expect oscillatory behavior of the solution.

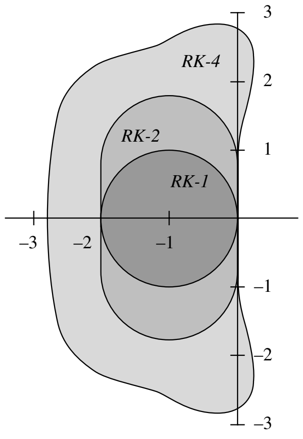

**Figure 12:** Stability domains for explicit Euler (*RK-1*), Heun (*RK-2*) and classical Runge-Kutta (*RK-4*) methods. If the complex valued number $\lambda h$ is within the shaded domain of a method, the algorithm is stable.

### 5.4.5 Order and Efficiency

The order of a method, meaning the power upon which the global error is dependent, is often but not always, equal to the number of steps or stages. The classical Runge-Kutta method is a four-stage method (the vector $\mathbf{F}(x,\mathbf{Y})$ of the differential equation is calculated four times) and has an error behavior of the fourth order. There are other methods, though, which display better behavior than might be expected based upon their stage. Collocation methods (implicit Runge-Kutta methods) which use the Lagrange-, Radau- and Lobatto points as collocation points are of this type. The global approximation error of the Radau method of stage $s$ has the error order $2s$-1 at the end of each integration step (element) and the order $s$+1 within the elements. The 3-stage collocation method shown in Fig. 10 (approximation with a polynomial of the third order) therefore has a 5th order global error.

The interesting thing about methods of a higher order is that they are generally more efficient than methods of a lower order. By efficiency we mean the number of calculations necessary to attain a determined accuracy of the result. Although the number of calculations necessary per integration step increases with each additional stage, the errors become smaller much faster. In theory, a method of stage $s$ with an error order $s$, is more efficient the higher $s$ is. The number of calculations necessary is proportional to the number of steps $N$ and to the stage $s$ of the method. On the other hand, the error is inversely proportional to the power $s$, resulting in

$$\log(error) = const - s\log(effort) \tag{28}$$

In a double logarithmic diagram, the error as a function of the effort decreases with a slope $s$. The larger the error order is, the steeper it is. One must be careful with this statement, though, because many practical factors can change the theoretical behavior in concrete problems.

## 5.5 ERROR ESTIMATION AND AUTOMATIC STEP CONTROL

An important point in the execution of numerical solutions of initial value problems is automatic step control. If the step-size is kept constant, $h$ must assume the smallest value necessary for the approximation in a range where the solution changes quickly. This also applies to areas of integration intervals where the solution changes more slowly. This leads to an often excessive number of calculations (not to mention the increase of rounding errors because of the many calculations).

An efficient algorithm needs an efficient automatic step control. This, in its turn, necessitates a dependable error estimate. In Section 5.2.4 we have only discussed a single approach: extrapolation of Euler's method. Now we are going to extend extrapolation to higher order methods. Then we will introduce so called embedded methods in which an error estimate is built into a Runge-Kutta algorithm. Examples of this are the Runge-Kutta-Merson, RK-Fehlberg and Dormand-Prince methods. Finally we shall discuss a step control mechanism in more detail.

### 5.5.1 Extrapolation

As described in Section 5.2.4, two solutions are computed over an interval: one with a single step of length $h$, the other having two steps of length $h/2$. Now, with a method having error order $p$, the local error is of order $p$+1. A derivation similar to the one that led to Eq. (10) yields

$$Y - y_1 = Ch^{p+1}$$
$$Y - y_2 = 2C\left(\frac{h}{2}\right)^{p+1}$$

which leads to the error estimate

$$Y - y_2 = \frac{y_2 - y_1}{2^p - 1} \tag{29}$$

Remember that this is the estimate for the local error. This is important for the step-size control idea discussed below. The extrapolated value for $y_2$ is

$$\hat{y}_2 = y_2 + \frac{y_2 - y_1}{2^p - 1} \tag{30}$$

### 5.5.2 Embedded Runge-Kutta Methods

There is an elegant method for error estimation which can be done in a single step. It is possible to construct methods which not only compute the (single-step) solution $y_1$ but also a second value $\hat{y}_1$ of higher order (usually $p$+1 if $p$ is the order of the method itself). In explicit Runge-Kutta methods this means that we construct an algorithm with coefficients from a matrix as in Table 7 with

$$y_1 = y_o + h\left(b_1 k_1 + \ldots + b_s k_s\right) \tag{31}$$

$$\hat{y}_1 = y_o + h\left(\hat{b}_1 k_1 + \ldots + \hat{b}_s k_s\right) \tag{32}$$

which are of order $p$ and $q$, respectively, where we usually have $q = p$+1. Note that the $k_i$ have to be calculated only once.

| | | | | | |
|---|---|---|---|---|---|
| 0 | | | | | |
| $c_2$ | $a_{21}$ | | | | |
| $c_3$ | $a_{31}$ | $a_{32}$ | | | |
| … | … | … | | | |
| $c_s$ | $a_{s1}$ | $a_{s2}$ | … | $a_{s,s-1}$ | |
| | $b_1$ | $b_2$ | … | $b_{s-1}$ | $b_s$ |
| | $\hat{b}_1$ | $\hat{b}_2$ | … | $\hat{b}_{s-1}$ | $\hat{b}_s$ |

**Table 7:** Matrix of coefficients of explicit Runge-Kutta (ERK) methods with embedded error estimator.

The error is then estimated to be equal to the difference of the two expressions

$$err = \left|\hat{y}_1 - y_1\right| \tag{33}$$

and can be used in a step-size control algorithm just like an error obtained from an extrapolation method.

### 5.5.3 A Step Control Algorithm

If a method is of error order $p$, the local error is of order $p$+1. This means that if we compare the error estimate *err* (obtained from Eq. (29) or Eq. (33)) to the specified tolerance *tol*, we should adjust the integration step $h$ to a new step length

$$h_{opt} = h\left(\frac{tol}{err}\right)^{1/(p+1)} \tag{34}$$

As before in Section 5.2.5, this can be applied to either the absolute or the relative error. In the latter case, *err* has to be divided by $y_1$.

In *Solving Ordinary Differential Equations I* by Hairer, Norsett and Wanner (p. 167), a description can be found how the algorithm (34) can be made safer. First, (34) is multiplied by a safety factor $f$ (such as 0.9 or 0.8), and factors $f_{\min}$ and $f_{\max}$ are introduced so as not to let the step change too fast. Then we can determine the new step from

$$h_{opt} = h\min\left(f_{\max}, \max\left(f_{\min}, f(tol/err)^{1/(p+1)}\right)\right) \tag{35}$$

In practice, a control method should be tried out for each numerical algorithm on some important examples before it is used. Good professional codes contain quite some additional details derived from long experience.

## 5.6 FINITE ELEMENT METHODS FOR ONE DIMENSIONAL BOUNDARY VALUE PROBLEMS

One-dimensional spatial problems lead to ordinary differential equation just like initial value problems in dynamical systems (Eq. (47) of Chapter 3). Again, nature usually offers the laws in the form of systems of first order equations:

$$\frac{d\mathbf{Y}}{dx} = \mathbf{F}(x, \mathbf{Y}) \quad , \quad \mathbf{LBC}(x_o, \mathbf{Y}_o) = 0 \quad , \quad \mathbf{RBC}(x_{end}, \mathbf{Y}_{end}) = 0 \tag{36}$$

The only—yet very important—difference to initial value problems is this: instead of initial values at one end of the interval of integration, commonly at $t$ = 0, we now have boundary values at both ends of the interval (Fig. 13): we speak of two point boundary value problems. If there are $n$ differential equations, there may be $n_l$ boundary conditions at the left end $x_o$, and $n_r = n - n_l$ boundary conditions at the right end $x_{end}$ of the interval of integration.

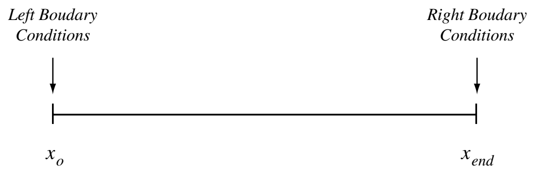

**Figure 13:** The range of the spatial independent variable $x$. The differential equations and the solutions are defined on the interval from $x_o$ to $x_{end}$. There are boundary values at both ends.

Having boundary conditions at both ends makes it impossible to solve the equations explicitly by starting at one end. (Actually, there are ways around this—exploited in so-called shooting methods—where one assumes values of the unknown functions at one end, integrates toward the other end, end verifies if the boundary conditions there are satisfied; if not, the guessed boundary values are adjusted, and the integration is repeated.) We always end up with an implicit set of equations made up of boundary value equations at one end, difference or finite element equations replacing the differential equations, and boundary equations at the other end. This requires solvers for (large) systems of possibly nonlinear equations.

If we have nonlinear differential equations of the general type as in Eq. (36), we also end up with nonlinear replacements after the approximation. Therefore, we either have to linearize the nonlinear approximation equations, or linearize the differential equations (and boundary equations). Let us assume we do the latter. We end up with

$$\frac{d\mathbf{Y}}{dx} = \mathbf{A}(x)\mathbf{Y} + \mathbf{B} \tag{37}$$

and appropriate boundary value equations. Here, **A** is an $n$x$n$ matrix, and **B** is an $n$ column vector. The $n$ vector **Y** represents the corrections to a guess of the solution (we need starting values for solving nonlinear equations by Newton-Raphson). Adding **Y** to the guess gives an improved solutions which may be used for a further round of integration for an even better approximation. For our following discussion it does not matter if we are starting with linear equations or if we have linearized differential equations. Therefore it is easiest to think in terms of linear boundary value problems from the start.

### 5.6.1 Collocation Finite Element Methods

In finite element approximations of (one-dimensional) boundary value problems one divides the interval of definition into $m$ sections called elements (Fig. 14). In each element, the unknown functions $Y_i$, $i = 1\ldots n$, are approximated by a usually quite simple function as in Eq. (21). Common examples of such functions are splines or polynomials. We shall discuss the use of polynomials here, because the resulting approximations are simple to formulate:

$$y(x) = \sum_{j=0}^{r} d_j x^j \tag{38}$$

The polynomial has a certain order $r$ for a given method, and the task is to find $r$+1 coefficients in each of the $m$ elements. This results in $(r+1)\cdot m\cdot n$ unknowns. Usually, we use $r$ approximations to the $n$ differential equations in $m$ intervals, leading to $r\cdot m\cdot n$ equations. These are augmented by $n$ boundary conditions and $(m–1)n$ conditions joining the end of the approximation (38) in one element to the beginning in the next element (continuity of the solution). This leads to a total of $(r+1)m\cdot n$ equations. However, we can satisfy the continuity condition by joining the piecewise functions at the end of the elements, leading to a reduced set of $(m\cdot r +1)\cdot n$ unknowns and equations.

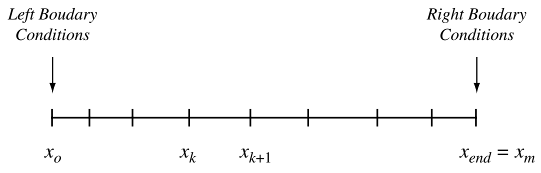

**Figure 14:** The range of the spatial independent variable $x$ is divided into $m$ elements. In each element the unknowns $Y$ are approximated by a simple function.

### 5.6.2 The System of Linear Equations

To be concrete let us discuss the case of two linear differential equations (with constant coefficients) of the form found in Eq. (37):

$$\begin{aligned} \frac{dY_1}{dt} &= a_{11}Y_1 + a_{12}Y_2 + b_1 \\ \frac{dY_2}{dt} &= a_{21}Y_1 + a_{22}Y_2 + b_2 \end{aligned} \tag{39}$$

which makes $n = 2$, supplemented by one boundary condition at both the left and right sides of the interval of definition. Furthermore, we assume four elements ($m = 4$), and a polynomial approximation of the second order ($r = 2$). According to what we just said, we should end up with $(4\cdot 2+1)\cdot 2 = 18$ linear equations to be solved. What is the structure of this system?

There are several preparatory steps that have to be taken before we arrive at a suitable form of the equations. First, we introduce a new variable $0 \le \xi \le 1$ on every interval (element) and replace the derivative *d/dt* by $d/d\xi$ (this introduces the factor $\Delta t$ = length of an element in each differential equation). Then we substitute the unknowns *y* at the points $\xi = 0$, 0.5, and 1 (0.5 and 1 shall be our collocation points) for the unknown coefficients $d_o$, $d_1$, $d_2$ in Eq. (38). Since we have two sets of *y*'s for the two differential equations, and three points (0, 0.5, 1) for every element, we end up with 6 unknowns and 2·2 = 4 collocation equations at $\xi = 0.5$ and 1 in every element. The matrix of this subsystem is shown in Table 8.

| $y_1(\xi = 0)$ | $y_2(\xi = 0)$ | $y_1(\xi = 0.5)$ | $y_2(\xi = 0.5)$ | $y_1(\xi = 1)$ | $y_2(\xi = 1)$ |
|---|---|---|---|---|---|
| $2/\Delta t$ | | $a_{11} - 2/\Delta t$ | $a_{12}$ | | |
| | $2/\Delta t$ | $a_{21}$ | $a_{22} - 2/\Delta t$ | | |
| $1/\Delta t$ | | | | $a_{11} - 1/\Delta t$ | $a_{12}$ |
| | $1/\Delta t$ | | | $a_{21}$ | $a_{22} - 1/\Delta t$ |

**Table 8:** Matrix entries for a single element of the $r = 2$ collocation finite element approximation.

Once the equations are assembled, the complete system of linear equations $\mathbf{C \cdot y = B}$ looks like Fig. 15. Efficient algorithms have to be written to solve such systems.

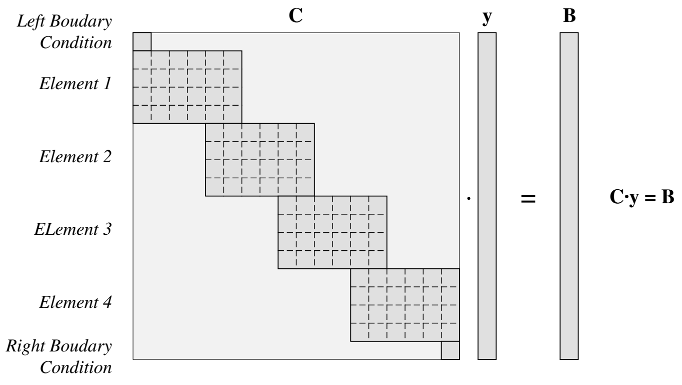

**Figure 15:** Form of the system of linear collocation and boundary equations for 4 elements and two equations. Nonzero elements are found only in the dark gray areas.