Chapter 2
Numerical Simulation of ODE Models

In the preceding chapter we had worked out how to establish possibly large ODE models for systems biological networks. In the present chapter, we deal with their numerical simulation. For this purpose, we describe various numerical integrators for initial value problems in necessary detail. In Sect. 2.1, we present basic concepts to characterize different discretization methods. We start with local versus global discretization errors, first in theory, then in algorithmic realization. Stability concepts for discretizations lead to an elementary pragmatic understanding of the term "stiffness" of ODE systems. In the remaining part of the chapter, different families of integrators such as Runge-Kutta methods, extrapolation methods, and multistep methods are characterized. From a practical point of view they are divided into explicit methods (Sect. 2.2), implicit methods (Sect. 2.3), and linearly implicit methods (Sect. 2.4), to be discussed in terms of their structural strengths and weaknesses. Finally, in Sect. 2.5, a roadmap of numerical methods is given together with two moderate problems that look rather similar, but require different numerical integrators. Moreover, we present a more elaborate example concerning the dynamics of tumor cells; therein we show, what kind of algorithmic decisions may influence the speed of computations.

2.1 Basic Concepts

Throughout this section we consider the *numerical integration* of in general nonlinear ODE initial value problems. For ease of writing, we confine our interest to autonomous problems

$$y' = f(y), \quad y(0) = y_0 , \tag{2.1}$$

unless explicitly stated otherwise. In Sect. 2.1.1, we explain two concepts of discretization errors at the simplest possible example, the classical Euler method. These concepts show up in the control of local discretization errors and their relation to the actually achieved final error, to be presented in Sect. 2.1.2. As the ODE systems arising from systems biology are typically large and "stiff", we derive stability concepts in Sect. 2.1.3 that help to understand this term. For practical applications, some rather pragmatic "definition" of stiffness is given in the final Sect. 2.1.4.

2.1.1 Local Versus Global Discretization Error: Theoretical Concepts

In this section, we discuss selected elementary discretization schemes in terms of the *timestep* or *step size* τ. We begin with the classical scheme discussed by Leonhard Euler even before the invention of the concept of differential equations.

Classical (Explicit) Euler Method

The first idea of this scheme is to discretize the ODE (2.1) by using the geometric *tangent* at the starting point

$$y_1 = y_0 + \tau f(y_0) \tag{2.2}$$

The second idea is to repeat the first step recursively as (see Fig. 2.1)

$$y_{k+1} = y_k + \tau f(y_k), \quad k = 0, 1, 2, \ldots. \tag{2.3}$$

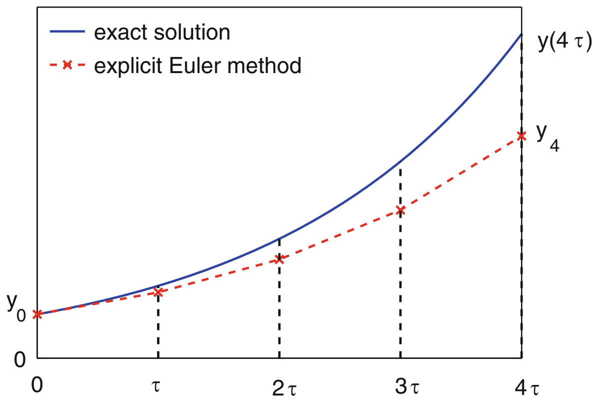


Fig. 2.1 Explicit Euler method with constant step size τ

For ease of presentation, we here stick to a *uniform* discretization with a single constant step size τ in this section. For a *non-uniform* discretization we should rather write

$$y_{k+1} = y_k + \tau_k f(y_k), \quad k = 0, 1, 2, \ldots . \tag{2.4}$$

Local Discretization Error

At the simple Euler discretization scheme we can already explain the general concepts of local and global discretization errors. After one discretization step, a deviation $y_1 - y(\tau)$ between the discrete and the continuous solution arises. In order to reveal the dependence of this deviation on τ, we use (2.1) in integrated form, i.e.

$$y(\tau) = y_0 + \int_0^\tau f(y(s))ds . \tag{2.5}$$

Then, together with (2.2), we are able to derive the relation

$$y_1 - y(\tau) = \tau f(y_0) - \int_0^\tau f(y(s))ds .$$

Taylor expansion of the above integrand yields

$$f(y(s)) = f(\underbrace{y(0)}_{y_0}) + s\underbrace{f_y(y_0)f(y_0)}_{=f'(y_0)} + \mathcal{O}(s^2) .$$

Here we have used the convenient notation $\mathcal{O}(s^p)$ for terms of order at least p. Upon inserting this expression into the integral above, the first right-hand term cancels and we arrive at

$$y_1 - y(\tau) = -\int_0^\tau [sf'(y_0) + \mathcal{O}(s^2)]ds = -\frac{\tau^2}{2}f'(y_0) + \int_0^\tau \mathcal{O}(s^2)ds .$$

As the integral term on the right-hand side is obviously $\mathcal{O}(\tau^3)$, we end up with a *local discretization error* estimate of the type

$$\|y_1 - y(\tau)\| \leq c \cdot \tau^2 , \tag{2.6}$$

where c is a constant containing information of the problem at hand (such as $f'(y_0)$).

Global Discretization Error

After N time steps according to (2.3) some *global discretization error* at *fixed* final time $T = N\tau$ will arise. Intuitively we obtain

$$\|y_N - y(T)\| \leq C \cdot N\tau^2 = C \cdot T\tau \ , \tag{2.7}$$

assuming $N \to \infty$ as $\tau \to 0$. Obviously, in the transition from local to global error we lose one order of τ.

It should be mentioned that the constants c in (2.6) and C in (2.7) are different. A rigorous proof of the result (2.7) can be found, e.g., in [16] and further references therein. As it turns out, such a proof shows that the above constant C is only bounded, if a *Lipschitz condition* of the kind

$$L\tau \leq \text{const} \tag{2.8}$$

holds, where L is the Lipschitz constant introduced in (1.31) and const means some small number, say, not much larger than 1. This is a severe *restriction on the step size* τ that holds for a large class of discretization methods and will come up at several occasions throughout this book.

Implicit Euler Method

Once the classical discretization (2.2) had been known, the question arose whether it might be useful to insert the tangent at the *new* value y_1 by virtue of

$$y_1 = y_0 + \tau f(y_1) \ . \tag{2.9}$$

Here the value y_1 is only *implicitly* defined by some in general nonlinear system of equations. That is why the above scheme is called *implicit* Euler discretization and accordingly (2.2) the *explicit* Euler discretization. Again the scheme is repeated recursively according to

$$y_{k+1} = y_k + \tau f(y_{k+1}), \quad k = 0, 1, 2, \ldots \ .$$

Formally speaking, estimates for the local and global discretization error will just repeat (2.6) and (2.7). The numerical solution of the algebraic equations (dimension d) per each time step is certainly much more work than the corresponding explicit Euler recursion. For so-called "stiff" problems, however, this additional computational amount pays off, see Sect. 2.1.4 below.

Implicit Trapezoidal and Midpoint Rule

Once the door had been opened to modify the classical Euler method by introducing the implicit structure, a symmetric right-hand side has been chosen such as

$$y_{k+1} = y_k + \frac{\tau}{2}\left(f(y_k) + f(y_{k+1})\right), \quad k = 0, 1, 2, \dots , \tag{2.10}$$

which is called the *implicit trapezoidal rule*. Another also symmetric modification is the *implicit midpoint rule*

$$y_{k+1} = y_k + \tau f\left(\frac{1}{2}(y_k + y_{k+1})\right), \quad k = 0, 1, 2, \dots , \tag{2.11}$$

We will come back to these elementary discretizations at various occasions.

General Case

The two introduced convergence concepts, even though merely exemplified at rather simple discretizaton schemes, directly carry over to more general discretization methods to be presented below in the remaining parts of Chap. 2. For so-called *one-step methods*, the local discretization error has the typical structure

$$\|y_1 - y(\tau)\| \le c_p \cdot \tau^{p+1} \tag{2.12}$$

in terms of some *order p* characteristic of the method, while the global error, again under some condition of the kind (2.8), satisfies

$$\|y_N - y(T)\| \le C_p \cdot N\tau^{p+1} = C_p \cdot T\tau^p \ . \tag{2.13}$$

Note that $p = 1$ holds for the explicit as well as for the implicit Euler discretization, while $p = 2$ holds for the implicit trapezoidal rule and the implicit midpoint rule; this can be directly derived from the *symmetry* $(y_k, y_{k+1}, \tau) \leftrightarrow (y_{k+1}, y_k, -\tau)$. In all cases, a reduction of τ implies a reduction of both the local and the global discretization error bounds.

Throughout the subsequent chapters we will restrict ourselves to the first discretization step from y_0 to y_1, but tacitly include the total step from y_0 to y_N via the successive recursions from y_k to y_{k+1} for $k = 0, 1, \dots$.

Step-Size and Order Control

Efficient modern integrators will adapt their performance to problem dependent information to choose "optimal" step sizes, say τ_k at step k, part of them also deliver locally "optimal" orders p_k. Note that not always the rule "the higher the order the better" holds, since the above constants c_p strongly influence the achievable step sizes. In summary, order and step-size control are linked – see, e.g., the material worked out in Sects. 2.2–2.4 for each of the discussed numerical integrators.

2.1.2 Local Versus Finally Achieved Accuracy: Algorithmic Concepts

Throughout this section, let a prescribed fixed integration interval $[0, T]$ be subdivided according to

$$0 = t_0 < t_1 < \ldots < t_N = T \ ,$$

i.e. by $N + 1$ integration points, chosen to be non-uniformly distributed for the time being. The local and global error concepts introduced in Sect. 2.1.1 above turn out to have their correspondence in the numerical realization. For ease of notation, we define the *local discretization errors* as $\delta y_j = y(t_j) - y_j$ and require, in terms of some suitable vector norm $\|\cdot\|$, that

$$\|\delta y_j\| \le \text{TOL} \ , \tag{2.14}$$

where TOL is a user prescribed *local error tolerance*. Such errors can be controlled by all modern numerical integrators, see Sects. 2.2–2.4 below. At time point t_j, let $\delta y(t_j)$ denote the *accumulated error*. At final time T one thus has the *global discretization error*

$$\|\delta y(T)\| = \text{ERR} \ , \tag{2.15}$$

wherein the value ERR usually can *not* be prescribed or controlled by numerical integrators. That is why it is of interest to study the relation between TOL and ERR in theoretical terms.

Error Propagation

In order to study the relation between local and global error, we derive a linearized theory of the error propagation, as illustrated in Fig. 2.2.
At some time step $t_j \to t_{j+1}$, we obtain the linearized relation

$$\delta y(t_{j+1}) \doteq \delta y_{j+1} + W(t_{j+1}, t_j)\delta y(t_j) \ , \tag{2.16}$$

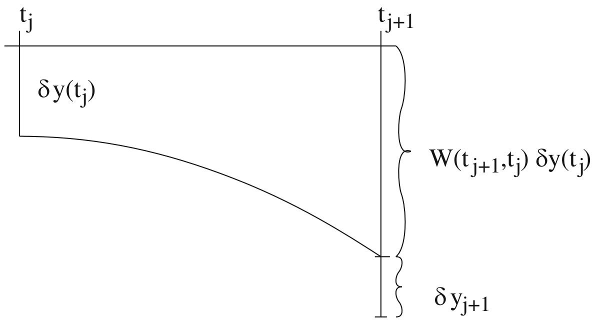


Fig. 2.2 Linearized local error propagation, see (2.16)

wherein $W(\cdot,\cdot)$ denotes the Wronskian matrix introduced in Sect. 1.3.2 above (for the non-autonomous case). By induction, starting with $\delta y_0 = 0$ and exploiting the semigroup property (1.37), here as $W(t_{j+2},t_j) = W(t_{j+2},t_{j+1})W(t_{j+1},t_j)$, we arrive at

$$\delta y(T) \doteq \sum_{j=1}^{N} W(T,t_j)\,\delta y_j \tag{2.17}$$

Taking norms, we get

$$\mathrm{ERR} \mathrel{\dot{\le}} \sum_{j=1}^{N} \|W(T,t_j)\| \cdot \overbrace{\|\delta y_j\|}^{\le \mathrm{TOL}} \le \mathrm{TOL} \sum_{j=1}^{N} \kappa[t_j,T] \tag{2.18}$$

in terms of the interval condition numbers $\kappa[\cdot,\cdot]$ introduced in (1.43) above. Since N depends on TOL, there is, in general, no *linear* relation between ERR and TOL.

Role of Interval Condition Number

Obviously, the key issue above is the structure of the interval condition number. If we apply the theoretical characterization in terms of the Lipschitz constant L defined in (1.31) and return, for simplicity, to a uniform grid with $t_j = j\tau$, we arrive at the theoretical estimate

$$\kappa[t_j,T] \le \exp(L(T-t_j)) = \exp(L(N-j)\tau) \tag{2.19}$$

and thus end up with the estimate

$$\mathrm{ERR} \mathrel{\dot{\le}} \mathrm{TOL} \cdot \frac{\exp(LT)-1}{\exp(L\tau)-1} \tag{2.20}$$

Such an estimate will only be reasonable for those ODE problems whose dynamical behavior can be characterized in terms of the Lipschitz constant; in Sect. 2.1.4 below, we will call such problems "non-stiff". Suppose, however, we have an ODE problem where

$$\kappa[t_j, T] \leq \text{const} \,, \tag{2.21}$$

which means that local errors die out asymptotically and thus dominate global errors. In mathematical analysis such problems are mostly called "dissipative", whereas in numerical analysis they are usually called "stiff". Insertion of (2.21) into (2.18) then yields

$$\text{ERR} \,\dot{\leq}\, \text{const} \cdot N \,\text{TOL} \,. \tag{2.22}$$

Total Error

Let the integration order p and the step size τ be fixed. Then, from (2.13), we roughly have the following global discretization error

$$\text{ERR} \;\doteq\; C_p T \tau^p \sim \tau^p \,.$$

Apart from the discretization errors, we will also have to deal with *rounding errors*. On a computer with relative precision eps ($\sim 10^{-16}$ typically today) we roughly obtain the contribution

$$\overline{\text{ERR}} \;\doteq\; \gamma_p N \text{eps} = \frac{\gamma_p \text{eps} T}{\tau} \sim \frac{1}{\tau}$$

with a constant γ_p depending on the discretization method. Note that for $\tau \to 0$ the discretization errors decrease, whereas the rounding errors increase, as shown schematically in Fig. 2.3. For the *total error* $\text{ERR}_{\text{total}} = \overline{\text{ERR}} + \text{ERR}$ there exists a smallest achievable accuracy $\text{ERR}_{\min}$ at τ_{opt}.

Summary

In numerical integrators only *local* accuracies can be controlled by the user prescribed error tolerance TOL. Global accuracy, say ERR, depends on the interval condition, say $\kappa[0, T]$, of the problem and can be tested by a few runs with different parameters TOL.

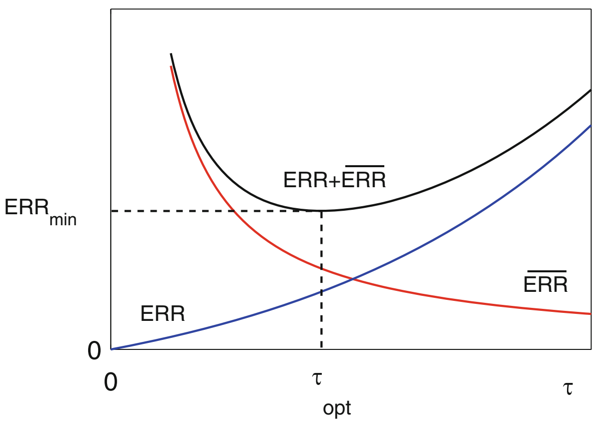


Fig. 2.3 Total error as sum of discretization error ERR and rounding error $\overline{\text{ERR}}$. The limit accuracy ERR$_{\text{min}}$ occurs at τ_{opt}

2.1.3 Stability Concepts for Discretizations

In Sect. 1.3.3 above, we had discussed the concept of asymptotic stability of ODE initial value problems. Here, we now deal with the question of whether the properties of the continuous problem are inherited to the discretized problems. In order to understand the subsequent line of argument, the reader may want to brush up his knowledge of the complex-valued exponential function by looking up Appendix A.1.

Dahlquist Test Model

For linear autonomous systems we had found out in Sect. 1.3.3 that they boil down to d simple scalar equations of the kind (1.51). That is why G. Dahlquist [12] had suggested in 1956 to study the stability properties of discretizations by the *test problem* (today named after him)

$$y' = \lambda y, \quad y(0) = 1, \quad \lambda \in \mathbb{C}\,. \tag{2.23}$$

The analytical solution of this problem is

$$y(\tau) = \exp(\lambda\tau) = \exp(z) \quad \text{where} \quad \tau \geq 0,\; z := \lambda\tau \in \mathbb{C}\,. \tag{2.24}$$

The trick here is to formulate the stability problem in terms of the complex-valued exponential function $\exp(z)$, which we discuss in some detail in Appendix A.1. From the relations (A.2) we may deduce the basic properties:

$$\begin{cases} |\exp(z)| \geq 1 & \Leftrightarrow \quad \Re(z) \geq 0 \\ |\exp(z)| \leq 1 & \Leftrightarrow \quad \Re(z) \leq 0 \\ |\exp(z)| = 1 & \Leftrightarrow \quad \Re(z) = 0 \end{cases} \tag{2.25}$$

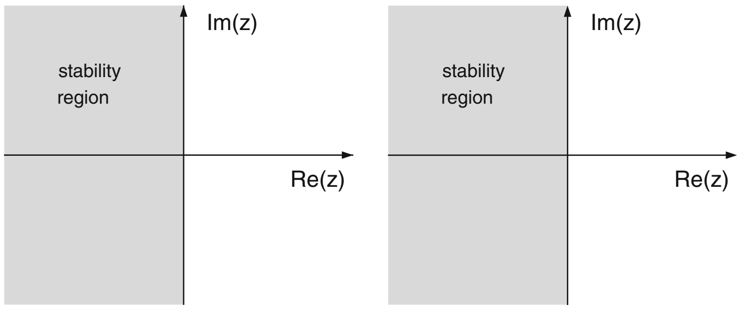


Fig. 2.4 Complex half-plane as stability region. *Left*: continuous solution. *Right*: discrete solution obtained by the implicit trapezoidal or the implicit midpoint rule

If we define the *stability region* by

$$\mathcal{S} := \{z \in \mathbb{C} : |y_1(z)| \le 1\}\,, \tag{2.26}$$

we may identify $\mathcal{S}$ for the continuous solution with the complex half-plane (see Fig. 2.4, left)

$$\mathbb{C}_- := \{z \in \mathbb{C} : \Re(z) \le 0\} \tag{2.27}$$

Examples of Stability Regions

For illustration, we apply the four elementary discretization schemes of Sect. 2.1.1 to the Dahlquist model (2.23). For the explicit Euler scheme we obtain

$$y_1 = y_0 + \tau f(y_0) = y_0 + \tau\lambda y_0 = 1 + \tau\lambda, \quad \Rightarrow \quad y_1 = 1 + z\,.$$

The corresponding stability region is shown in Fig. 2.5, left. In a similar way we get for the implicit Euler scheme

$$y_1 = y_0 + \tau f(y_1) = y_0 + \tau\lambda y_1 = 1 + \tau\lambda y_1, \quad \Rightarrow \quad y_1 = \frac{1}{1-z}\,.$$

The corresponding stability region is shown in Fig. 2.5, right. Finally, for the implicit trapezoidal and the implicit midpoint rule, which are equivalent for linear problems, we get

$$y_1 = y_0 + \tau(f(y_1) + f(y_0))/2 = y_0 + \tau\lambda(y_1 + y_0)/2 \quad \Rightarrow \quad y_1 = \frac{1 + z/2}{1 - z/2}\,,$$

The corresponding stability region is shown in Fig. 2.4, right.

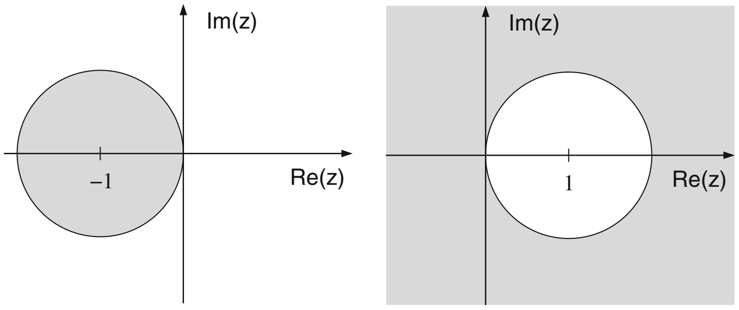


Fig. 2.5 Stability regions. *Left*: explicit Euler scheme. *Right*: implicit Euler scheme ("superstability")

From Fig. 2.4, one might think that the discretizations based on the implicit trapezoidal rule or the implicit midpoint rule are best, since they perfectly inherit the stability region from the continuous solution. However, the continuous solution has an additional desirable feature: For $z \to \infty$ one obtains

$$|\exp(z)| \quad \longrightarrow \quad \begin{cases} \infty, & \text{if } \Re(z) > 0 \\ 1, & \text{if } \Re(z) = 0 \\ 0, & \text{if } \Re(z) < 0 \end{cases}$$

If we compare this limit property for the elementary discretizations, we see that

$$|y_1(z)| \quad \longrightarrow \quad \begin{cases} \infty, & \text{for the explicit Euler scheme} \\ 1, & \text{for implicit trapezoidal or midpoint rule} \\ 0, & \text{for the implicit Euler scheme} \end{cases}$$

From (A.3) we know that the limit $z \to \infty$ for the complex exponential function depends on the path taken to approach this point. Such a behavior cannot be mimicked by polynomials or rational functions, where a unique limit independent of the path of approach exists. Consequently, we cannot expect to be able to realize all of the properties of the analytical solution by a single discretization. In view of this insight, several stability concepts have been introduced to characterize desirable features of different discretizations.

A-Stability

In view of the fact that $\mathbb{C}_- = \mathcal{S}$ for the continuous solution, this concept is defined by requiring the weaker property

$$\mathbb{C}_- \subset \mathcal{S}$$

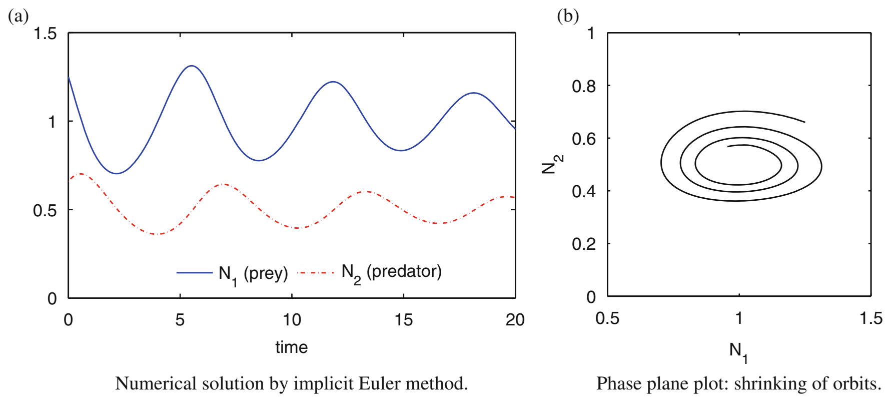


Fig. 2.6 *Example of superstability:* Predator-prey model (1.10) as in Fig. 1.2. Qualitatively wrong solution by implicit Euler method with constant step size $\tau = 0.1$. Correct numerical solution see Fig. 1.2

for discretizations. Looking at the above Figs. 2.4 and 2.5, we directly see that the explicit Euler scheme is *not* A-stable, whereas the other three discretizations are. Note, however, that the implicit Euler scheme has the undesirable feature to supply decaying solutions even when the continuous solutions increase – associated with those parts of $\mathcal{S}$ that cover part of the positive complex half-plane. This unwanted property is called *superstability* and illustrated for the predator-prey model in Fig. 2.6.

A(α)-Stability

As it turns out, some discretizations exhibit only "almost" A-stability. In order to quantify this feature, O. Widlund [62] in 1967 introduced some angle domain (shown in Fig. 2.7)

$$\mathbb{C}(\alpha) := \{z \in \mathbb{C}\,;\ |\arg(z)| \leq \alpha\}$$

and suggested the weakened concept of A(α)-stability defined via

$$\mathbb{C}(\alpha) \subset \mathcal{S}\ .$$

Observe that $\mathbb{C}(\alpha)$ permits a characterization of the complex value $z = \lambda t$, which holds, given λ, *for all* $t > 0$.

Clearly, since $\mathbb{C}(\pi/2) = \mathbb{C}_-$, it is tacitly understood that the larger α, the "more stable" the method is. Such a statement, however, should be taken with care in view of the limit feature mentioned above.

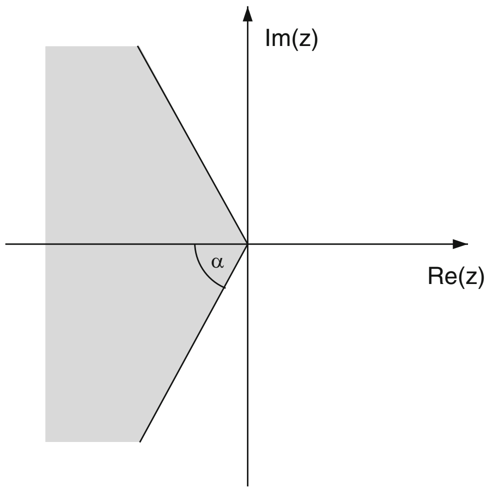


Fig. 2.7 Angle domain $\mathbb{C}(\alpha)$ for the definition of weakened stability concepts

L-Stability

This stability concept dates back to B.L. Ehle [27] in 1969. It additionally incorporates the asymptotic behavior of a discretization scheme for $z \to \infty$ by requiring

$$\mathbb{C}_- \subset \mathcal{S} \text{ and } y_1(\infty) = 0 \,. \tag{2.28}$$

In other words: A discretization method is L-stable, if it is A-stable and vanishes at $z = \infty$. Looking again back to our four sample discretizations, we observe: The explicit Euler scheme is *not* L-stable, since it satisfies neither of the two conditions. The implicit Euler scheme is L-stable, satisfying both conditions (but exhibits the unwanted superstability!). The implicit trapezoidal and the implicit midpoint rule are *not* L-stable, since $|y_1(\infty)| = 1$, which violates the second condition above. Even worse, the discrete values y_k oscillate according to

$$y_k \quad \to \quad (-1)^k \quad \text{as} \quad z \to \infty \,.$$

This asymptotic behavior is consistent with the behavior (A.3) of the continuous solution along the imaginary axis. It already shows up “close to” the imaginary axis for “large” $\Im z$ as $k \to \infty$ via the occurrence of “spurious” oscillations, i.e. via “numerical artifacts” that have may have nothing to do with the actual solution, see Fig. 2.8. For this reason, these two discretizations should only be applied, if the problem itself has eigenvalues close to the imaginary axis (and thus the solution exhibits “real” oscillations). As a rule of thumb, whenever a problem is nonlinear, then the implicit midpoint rule should be preferred over the implicit trapezoidal rule – for reasons worked out, e.g., in [16, Section 6.3.2] and references therein.

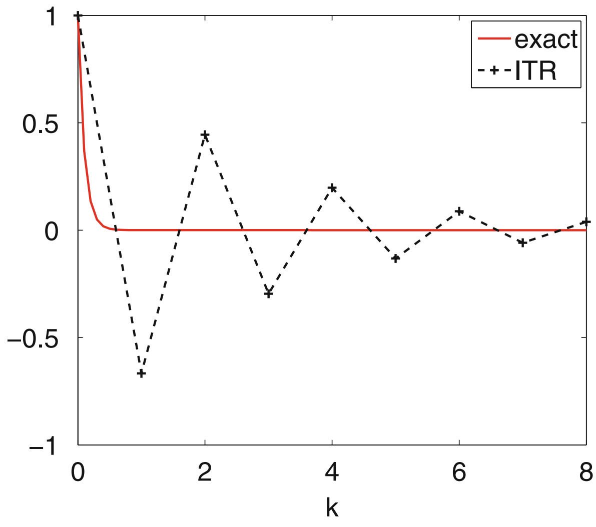


Fig. 2.8 Oscillatory behavior of the discrete values y_k obtained via the implicit trapezoidal rule with constant stepsize $\tau = 1$ applied to Dahlquists's test equation with $\lambda = -10$

L(α)-Stability

Just like A(α)-stability, a stability concept weakening L-stability exists as well, accordingly called L(α)-stability and defined via

$$\mathbb{C}(\alpha) \subset \mathcal{S} \text{ and } y_1(\infty) = 0 \; .$$

Again: the larger α, the better. Note, however, that the inclusion of the asymptotic property $y_1(\infty) = 0$ makes this concept much more useful in assessing numerical discretization schemes than mere A(α)-stability. We will return to this concept repeatedly.

2.1.4 *Stiffness of ODE Problems*

The stability analysis of the preceding Sect. 2.1.3 has been based on the simple Dahlquist test model,

$$y' = \lambda y, \quad y(0) = 1 \; ,$$

where $\lambda \in \mathbb{C}$ plays a rather different role depending on whether the sign of $\Re\lambda$ is positive or negative. However, if we calculate the Lipschitz constant L defined in (1.31), which is essential for all proofs of uniqueness of ODE models, we obtain

$$L = |\lambda| \; .$$

Hereby, the essential sign information is lost. Upon recalling the relation (2.8), we come up with some step-size restriction of the kind

$$\tau \mathrel{\dot{\le}} \frac{1}{|\lambda|}$$

Things appear differently when seen through the glass of the *pointwise condition number* defined in (1.42): For the Dahlquist model we get

$$\kappa_0(t) = \exp(\Re\lambda t)\ ,$$

which clearly maintains the necessary information of the sign of $\Re\lambda$, as can be seen in (2.25). Note that the *interval condition number* defined in (1.43) arises as

$$\kappa[0,t] = \begin{cases} \kappa_0(t), & \text{if } \Re(\lambda) > 0 \\ 1, & \text{if } \Re(\lambda) \le 0 \end{cases}$$

In view of (2.18), we now see that

$$\text{ERR} \mathrel{\dot{\le}} \text{TOL} \cdot \begin{cases} \dfrac{\exp^{\Re\lambda T} - 1}{\exp^{\Re\lambda\tau} - 1}, & \text{if } \Re(\lambda) > 0 \\ N, & \text{if } \Re(\lambda) \le 0 \end{cases}$$

The above factor N may reduce to essentially around 1, if the exponent is "large negative", which is the case in strongly dissipative problems, where the global error is equivalent to the local error. If we turn from the analytic solution of the Dahlquist model to a discrete solution obtained by some selected discretization scheme, then any *step-size restrictions* will show up via the condition $z = \lambda\tau \in \mathcal{S}$ in terms of the stability region $\mathcal{S}$. If, in addition, the stability region nicely models the stability region of the analytic solution, then such a constraint will be reasonable. So we fall back on the stability concepts of the Sect. 2.1.3. Summarizing, discretization methods that mimic the stability behavior of the analytic solution of the Dahlquist model will have a tolerable error propagation, whereas others do not.

Characterization of Stiffness

There are countless numbers of definitions of "stiffness" of ODE problems, see, e.g., the textbook [16] and references therein. For computational scientists that are not mathematicians, we offer the following rather pragmatic definition:
For stiff ODE problems, the additional computational amount of solving linear or nonlinear equations within each (linearly) implicit discretization step pays off, since the number of discretization steps is significantly reduced.

Note that this "definition" is rather ad hoc and will depend on the required accuracy and the selected discretization method. Practically speaking, if the nature of the problem at hand is not clear in advance, one might start with an explicit discretization; if the problem can be handled with "reasonably looking" computational step sizes, then we regard it to be non-stiff. If, however, "too strong" step-size restrictions occur, so that "too many" time steps need to be taken, see the theoretical restrictions (2.8) or (2.46), then we regard the problem to be stiff. Note that any arising small step sizes would be appropriate in the *transition phase* of an otherwise stiff problem, but awkward in the *stationary phase*. Upon observing such an undesirable behavior one will switch to a (linearly) implicit discretization method.

For a *qualitative* characterization of a given problem as *stiff*, one will examine whether the trajectory asymptotically approaches some *equilibrium point* or some *stationary smooth* solution. In this sense, most of the ODE models arising in systems biology (including all of the problems mentioned in Sect. 1.2) are stiff and thus deserve the numerical solution by some (linearly) implicit discretization method.

In order to develop some *geometric intuition*, we recall Fig. 1.9 above, where we had illustrated *stability* versus *inherent instability* around some more general stationary solution $g(t)$. For ease of understanding, let us again characterize these two cases:

- An asymptotically *unstable* or "non-stiff" problem (Fig. 1.9, right) should be attacked by some *explicit* numerical integrator, which will automatically choose small step sizes τ that are totally in order, since they reflect the dynamics of the problem.
- An asymptotically *stable* or "stiff" problem (Fig. 1.9, left) should be attacked by some *(linearly) implicit* numerical integrator, which will choose large step sizes τ that reflect the lack of dynamics of the stationary solution $g(t)$ at the computational expense of solving nonlinear systems in each discretization step; if any explicit numerical integrator were chosen in this case, extremely small step sizes would automatically be selected that would blow up computing times beyond any tolerable amount.

2.2 Explicit Numerical Integration Methods

In this section, we present three extensions of the *explicit* Euler method. The first two ones are *one-step methods*, i.e. methods that only use information from the present step to compute approximations for the next step; these include Runge-Kutta methods in Sect. 2.2.1 (dating back to 1895) and extrapolation methods in Sect. 2.2.2 (with origins back in 1910). The third one is a *multistep method*, i.e. a method that exploits the history of a trajectory to compute the next step. Among them we restrict our attention to Adams methods (dating back to 1855). Even though all of these methods seem to be rather old, there are modern *adaptive* versions that should definitely be preferred for the numerical simulation of ODE models from systems biology.

2.2.1 Runge-Kutta Methods

Already in 1895, C. Runge suggested an improvement over the classical Euler scheme, which read

$$y_1 = y_0 + \tau f\left(y_0 + \frac{\tau}{2} f(y_0)\right)$$

Careful examination of this scheme reveals an order $p = 2$, compare definition (2.12) above, i.e. one order higher than that of the explicit Euler scheme. This increase of order has been attained by introducing an explicit Euler step with $\tau/2$ as step size inside $f(\cdot)$. In 1901, W. Kutta recognized the general nested structure to increase the order: He suggested what nowadays is called an *s-stage explicit Runge-Kutta scheme*

$$\begin{aligned} k_i &= f\left(y_0 + \tau \sum_{j=1}^{i-1} a_{ij} k_j\right), \quad i = 1, \ldots, s \\ y_1 &= y_0 + \tau \sum_{i=1}^{s} b_i k_i \end{aligned}$$

Note the key feature that this scheme is recursive in the unknown directions k_i. Any Runge-Kutta method can be characterized by the coefficients (b_i), (a_{ij}), in matrix vector notation written as $(b, \mathcal{A})$, where b is an s-vector and $\mathcal{A}$ a lower triangular (s, s)-matrix with zero diagonal entries – which represents the recursive structure of the scheme.

Remark 5 When applied to *non-autonomous* systems with $f(t, y)$ as right-hand sides, the above terms are to be replaced by terms of the kind

$$f\left(t_0 + c_i\tau, y_0 + \tau \sum_{j=1}^{i-1} a_{ij} k_j\right) \quad \text{where} \quad c_i = \sum_{j=1}^{i-1} a_{ij} \, . \tag{2.29}$$

Once he had detected the general structure, W. Kutta also worked out a rather economic scheme of order $p = 4$, today named as the *classical* RK4 scheme:

$$\begin{aligned} k_1 &= f(y_0) \\ k_2 &= f(y_0 + \frac{\tau}{2} k_1) \end{aligned}$$

$$\begin{aligned}
k_3 &= f(y_0 + \frac{\tau}{2}k_2) \\
k_4 &= f(y_0 + \tau k_3) \\
y_1 &= y_0 + \tau\left(\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4\right)
\end{aligned}$$

Surprisingly, this ancient scheme is today still seen in modern computational science! However, in the meantime of more than 100 years, much more efficient higher order RK schemes have been worked out – see below.

Error Estimation and Step-Size Control

In order to simulate an ODE model reliably, the discretization error must be kept below a prescribed accuracy threshold. As worked out in Sect. 2.1.1, we have to deal with both local and global discretization errors. An estimation of the global discretization error, which means the finally achieved accuracy, would require an unreasonably large computational amount. The local discretization error, however, can be estimated via the construction of so-called *embedded* Runge-Kutta methods. In this approach, two RK methods with common coefficients $\mathcal{A}$, but different coefficients b and $\hat{b}$ are combined so that the scheme with b has order p, say, while the one with $\hat{b}$ has order $p-1$. After one step from $t = 0$ to $t = \tau$, the discretization error can be approximated according to

$$\|y_1 - y(\tau)\| \approx \|\sum_{i=1}^{s} \tau(b_i - \hat{b}_i)k_i\| =: \epsilon \tag{2.30}$$

A number of subtle considerations is necessary to back this device theoretically, see, e.g., [16, Section 5.3].

If the error of the higher order RK method with coefficients $(b, \mathcal{A})$ is estimated, the corresponding embedded RK method is usually written as RK (p-1)p, say RK (7)8. In this case, formula (2.12) combined with (2.30) above yields

$$\epsilon \approx \|y_1 - y(\tau)\| \le c_p \tau^{p+1} . \tag{2.31}$$

Suppose now we want to identify an "optimal" step size τ^* such that

$$\epsilon^* \approx \|y_1 - y(\tau^*)\| \doteq c_p(\tau^*)^{p+1} \le \text{TOL} ,$$

where TOL is a user prescribed local error tolerance. Division of the two relations then leads to the formula

$$\tau^* = \tau \sqrt[p+1]{\frac{\rho \cdot \text{TOL}}{\epsilon}} \tag{2.32}$$

where we have introduced some safety parameter $\rho < 1$, usually $\rho = 0.9$, with the aim to (roughly) ensure that the thus selected step size would then actually lead to

$$\epsilon^* \leq \mathrm{TOL}\ .$$

If several RK methods of different orders are realized, then the whole device can be enriched by some *order control*, details of which we will present in Sect. 2.2.2 below in the context of some subset of RK methods.

Dense Output

If more output data are wanted than are supplied by the step-size control, then the idea of an additional embedded RK method is again exploited; generalizing (2.31) one requires that

$$\|y_1(\theta) - y(\theta\tau)\| \leq c_{\bar{p}}\tau^{\bar{p}+1} \quad \text{for} \quad \theta \in [0, 1]\ . \tag{2.33}$$

The art is to find formulas that yield an order $\bar{p}$ as high as possible. Efficient RK methods usually include such a device, which also permits an extension to the numerical treatment of delay differential equations; as an example we mention the code RETARD due to E. Hairer.

Dormand-Prince Integrators

In recent decades, a whole "RK technology" for the construction of higher order RK methods has emerged, see, e.g., [16, Section 4.2.2] and references therein to the original literature. Given stage numbers s, the general principle is to determine the coefficients $(b, \mathcal{A})$ such that the discretization errors are of prescribed order p. This leads to a set of *(underdetermined) nonlinear algebraic* equations, rapidly growing with increasing orders, see Table 2.1. Note that the number of conditions only depends on the order p, not on the number s of stages, which only influences the number $(s^2 + 1)/2$ of independent coefficients to be determined. To solve these equations, additional wishes may be fulfilled concerning, e.g., embedding with more than two combined RK methods, economy of function evaluations as well as reliability and robustness of step-size control devices.

Table 2.1 Number N_p of algebraic equations for coefficients of Runge-Kutta methods depending on order p

p	1	2	3	4	5	6	7	8	9	10	20
N_p	1	2	4	8	17	37	85	200	486	1 205	20 247 374

Starting around 1980, J. R. Dormand and P. J. Prince [26] have developed a sequence of highly efficient explicit Runge-Kutta methods up to higher order, putting all theoretical and algorithmic pieces together. Their presently most efficient codes DOPRI5 and DOP853 have been economized with respect to number of function evaluations, efficiency of step-size control, dense output etc. The code DOP853 due to E. Hairer additionally realizes an automatic control of orders among the embedded orders $\{8, 5, 3\}$.

2.2.2 *Extrapolation Methods*

In this section, we present certain discretization methods that formally are a subset of explicit Runge-Kutta methods, but are constructed in a rather different way, along which the cumbersome solution of the many algebraic equations, see Table 2.1, can be avoided. For a general survey on extrapolation methods for (non-stiff and stiff) ODE problems we refer to [14].

Basic Procedure

The general idea of extrapolation methods is to run a simple *basic discretization* with *successively reduced internal step sizes*

$$\sigma_1 = \tau/n_1 > \sigma_2 = \tau/n_2 > \ldots \quad \text{for given} \quad 1 \le n_1 < n_2 < \ldots \in \mathcal{F}$$

up to the next time point τ. In this way, successively "better" approximations $y_1(\tau)$ emerge that can be regarded as functions of σ and written as $y_1(\tau;\sigma_1), y_1(\tau;\sigma_2), \ldots$. These approximations serve as nodal values for polynomial interpolation over the nodes $[\sigma_1, \ldots, \sigma_\nu]$. The interpolation polynomials $p(\sigma|\sigma_1, \ldots, \sigma_\nu)$ are then evaluated at $\sigma = 0$, which lies outside the interpolation interval, hence the name *extra*polation, see Fig. 2.9.

Theoretical basis for such a discretization is the existence of some "polynomial-like" expansion

$$y_1(\tau;\sigma) - y(\tau) = g_1(\tau)\sigma^{\gamma_1} + g_2(\tau)\sigma^{\gamma_2} + g_3(\tau)\sigma^{\gamma_3} + \ldots$$

A closer examination of such expansions for a class of discretization methods reveals that they would not converge, but can be replaced by *asymptotic expansions* of the kind

$$y_1(\tau;\sigma) - y(\tau) = \sum_{j=1}^{N} g_j(\tau)\sigma^{\gamma_j} + G_{N+1}(\tau;\sigma)\sigma^{\gamma_{N+1}} \,, \tag{2.34}$$

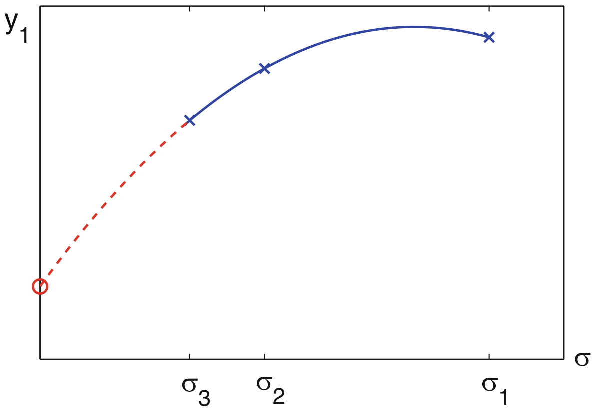


Fig. 2.9 Idea of extrapolation: Evaluation of the interpolation polynomial $p(\sigma|\sigma_1, \sigma_2, \sigma_3)$ at the limit $\sigma = 0$

Table 2.2 Aitken-Neville scheme for extrapolation methods (here $k = 3$)

$$\begin{tikzcd} T_{11}y \arrow[dr] & & \\ T_{21}y \arrow[r] \arrow[dr] & T_{22} \arrow[dr] & \\ T_{31}y \arrow[r] & T_{32} \arrow[r] & T_{33} \end{tikzcd}$$

where a remainder term G takes care of the convergence properties. These expansions serve the purpose needed for extrapolation methods. Proofs of their existence including the boundedness of the remained terms are quite subtle and can be found in the usual textbooks, see, e.g., [16, Section 4.3] and references therein.

Explicit Euler Discretization

Let us explain the method at the simplest possible example, when the explicit Euler discretization is selected as the basic discretization. Starting with $\eta_0 = y_0$ the basic scheme for internal step size $\sigma_\nu = \tau/n_\nu$ reads

$$\eta_{n+1} = \eta_n + \sigma_\nu f(\eta_n), \quad n = 0, \ldots, n_\nu - 1, \quad \rightarrow \quad y_1(\tau; \sigma_\nu) = \eta_{n_\nu} \ . \tag{2.35}$$

Here the numbers n_ν are chosen from some *subdivision sequence* $\mathcal{F}$. For this basic scheme, the existence of an asymptotic expansion (2.34) with $\gamma_j = j$ can be shown. We thus may extrapolate to the limit $\sigma = 0$ using the Aitken-Neville algorithm (e.g.,[17, Section 7.1.2]). Starting with $y_1(\tau; \sigma_i) = T_{i,1}$, this scheme reads in our case

$$T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\frac{n_i}{n_{i-k+1}} - 1} \tag{2.36}$$

For an illustration see Table 2.2.

For the discretization error after k extrapolation steps one obtains

$$\varepsilon_{ik} := \|T_{i,k} - y(\tau)\| \doteq \|g_k'(0)\|\tau\sigma_i \cdots \sigma_{i-k+1} = \gamma_{ik}\|g_k'(0)\|\tau^{k+1} \ , \tag{2.37}$$

with the constant

$$\gamma_{ik} = (n_{i-k+1} \cdot n_i)^{-1} \ .$$

Along these lines, an extrapolation method can be designed. But there exists a "better" option to be described next.

Remark 6 Formally, this extrapolation method is a special explicit RK method with stage number

$$s = \sum_{\nu=1}^{k} n_\nu \ ,$$

where k is the maximum index in the extrapolation table. Its minimum as achieved for the *harmonic* subdivision sequence

$$\mathcal{F}_H = \{1, 2, 3, 4, \ldots\}$$

The present codes actually realize this sequence.

Explicit Midpoint Rule

Already in 1910, C. Richardson had suggested to apply τ^2-extrapolation by exploiting the *symmetry* of the explicit mid-point rule

$$\eta_{n+1} = \eta_{n-1} + 2\sigma_\nu f(\eta_n) \ , \quad n = 1, 2, \ldots \ .$$

Obviously, this recurrence cannot be realized without some value for η_1. In other words: it needs a *starting step*, a problem that Richardson had not been able to solve.

It took until 1963 that W. B. Gragg in his thesis [29] was able to prove that a simple explicit Euler starting step is enough to guarantee the existence of a *quadratic* asymptotic expansion (2.34) with $\gamma_j = 2j$. Moreover, he showed that

- there exist *two different* quadratic asymptotic expansions for n_ν either *even* or *odd* and
- for an *even* subdivision sequence one additionally gains one order in τ in the approximation error.

Gragg also suggested the special final step

$$y_1(\tau;\sigma_\nu) := \overline{\eta}_n = \frac{1}{4}(\eta_{n-1} + 2\eta_n + \eta_{n+1}), \quad n = n_\nu \text{ even} . \tag{2.38}$$

This final step, which requires the additional evaluation of $f(\eta_n)$, is not really crucial, but useful for dense output, see below. Again this method is formally an explicit RK method, so that minimizing the stage number leads one to implement the *double harmonic* subdivision sequence (see [13])

$$n_\nu \in \mathcal{F}_{2H} = \{2, 4, 6, 8, \ldots\ldots\} .$$

Quadratic extrapolation. On this basis, one just needs to modify the Aitken-Neville algorithm (2.36) in the form

$$T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{n_i}{n_{i-k+1}}\right)^2 - 1} . \tag{2.39}$$

Again one gets a triangular extrapolation tableau as shown in Table 2.2. For the discretization error after k extrapolation steps one here obtains

$$\varepsilon_{ik} := \|T_{i,k} - y(\tau)\| \doteq \|g_k'(0)\|\tau\sigma_i^2 \cdots \sigma_{i-k+1}^2 = \gamma_{ik}^2\|g_k'(0)\|\tau^{2k+1} , \tag{2.40}$$

with the same constant γ_{ik} as defined in (2.37).

Order and Step-Size Control

In order to characterize the linear versus quadratic asymptotic expansions, we write $\gamma_j = \gamma j$ with $\gamma = 1$ for the explicit Euler scheme and $\gamma = 2$ for the explicit mid-point rule. On the basis of the error formulas (2.37) and (2.40), the above formula (2.32) readily suggests "optimal" step sizes for each column index k in Table 2.2

$$\tau_k^* = \tau \sqrt[\gamma k+1]{\frac{\rho \cdot \mathrm{TOL}}{\epsilon_{k+1,k}}}, \quad \rho < 1 . \tag{2.41}$$

For strong algorithmic reasons one uses the *subdiagonal* error estimate

$$\epsilon_{k+1,k} := \|T_{k+1,k} - T_{k+1,k+1}\| \approx \|T_{k+1,k} - y(\tau)\|$$

Thus each column is associated with its own step size suggestion. As for the amount of work, this is essentially only dependent on the row i in the tableau, so that we may denote it by A_i. In order to measure the *work per unit step* we define the computationally available quantities

$$W_k := \frac{A_{k+1}}{\tau_k}\tau = A_{k+1}\left(\frac{\epsilon_{k+1,k}}{\mathrm{TOL}}\right)^{\frac{1}{\gamma_{k+1}}} .$$

With this set of values, we may find an "optimal" column index, say q, by requiring the computable criterion

$$W_q = \min_{k=1,\ldots k_{\max}} W_k \tag{2.42}$$

Here a "greedy algorithm" argument has been used, see [17]. Once q is determined, the step size $\tau^* = \tau_q^*$ is taken as the basic step size for the next step. (In passing we note that the same kind of idea is used to select an "optimal" order within embedded explicit RK codes like DOP853.)

In addition to this order and step-size selection, the whole approximation table T_{ik} is exploited to monitor any unwanted deviation from an expected convergence pattern, details are left to [14]. This abundance of information makes extrapolation methods extremely robust in real life applications.

Dense Output

In general, extrapolation integrators adaptively select rather large step sizes due to their efficient order and step-size control. If more output data are wanted, then an extra tool is necessary. In connection with the explicit Euler or midpoint rule, it is easy to add a Hermite interpolation tool, see [17, Section 7.1.2]. This cubic interpolation is based on the information $y_0, f(y_0)$ and $y_1, f(y_1)$. The discrete values y_1 are anyway available at the end of each τ-step; in order to get some accurate value $f(y_1)$, one will evaluate f after the extrapolation for y_1; this does not increase the overall computational amount, since this value can be stored to be used as starting value for the next step. An efficient higher order technique for dense output in explicit extrapolation methods has been work out by E. Hairer and A. Ostermann in [32]; for the explicit midpoint rule, this technique requires a change of the subdivision sequence away from $\mathcal{F}_{2H}$, which is why we here do not pursue it. For the purpose of systems biology, we are only interested to get a cheap way of evaluating data at prescribed dense output points – an aim that can be achieved to sufficient accuracy already with the cubic Hermite interpolation, given the adaptive step-size and order control.

Note, however, that along this line backward dense output information is available only *after* each τ-step; consequently, such a tool is not sufficient to extend the extrapolation method in the direction of delay differential equations, since there the interpolation values are necessary *during* the discretization.

Explicit Extrapolation Codes

An extrapolation code EULEX based on the explicit Euler scheme as basic discretization has been exemplified in [14]; this code essentially served as a model to explain the idea of an extrapolation method. Based on the explicit mid-point rule, two implementations are quite popular, DIFEX1 due to [14], which includes the above explained dense output option, and ODEX due to [34]; the two codes differ only slightly in secondary details of the above order and step-size control and are similarly efficient, both of them typically twice as fast as EULEX.

2.2.3 Adams Methods

In the two previous Sects. 2.2.1 and 2.2.2 we had presented one-step methods, i.e. methods that only use information from $t_0 = 0$ to compute an approximation y_1 at the next time step $t_1 = \tau$. Already in 1855, J. C. Adams had the idea to exploit more of the "history" of the trajectory.

General Scheme

Adams directly started from the integral form (2.5), which we repeat for convenience as

$$y(\tau) - y_0 = \int_{t=0}^{\tau} f(y(t))\, dt\,.$$

The idea is to replace the integrand f by a polynomial g_k interpolating the $k+1$ "previous" values

$$f(y_1), f(y_0), f(y_{-1}), \ldots, f(y_{-k+1})\,.$$

This interpolation polynomial is unique and so we may write

$$f(y(t)) \quad \rightarrow \quad g_k(t) = \sum_{j=0}^{k} L_j(t) f(y_{j-k+1})$$

where the L_j denote the Lagrange polynomials that depend on the interpolation nodes only, see, e.g., [17, Section 7.1]. Insertion of this expression yields a discretization scheme of the kind

$$y_1 - y_0 = \tau\,(\beta_k f(y_1) + \beta_{k-1} f(y_0) + \beta_{k-2} f(y_{-1}) + \ldots + \beta_0 f(y_{k-1})) \qquad (2.43)$$

where the coefficients $\beta_k, \ldots, \beta_0$ are uniquely determined by

$$\beta_j = \frac{1}{\tau} \int_{t=0}^{\tau} L_j(t)\, dt = \int_{\theta=0}^{1} L_j(\theta\tau)\, d\theta \tag{2.44}$$

If $\beta_k = 0$, the scheme is *explicit*, in which case the polynomial g_k is of order k.

Remark 7 For readers with theoretical interest we briefly want to mention that the above Adams method is not just an arbitrary candidate out of a large class of possible multistep methods, but has an outstanding *stability* property. This stability corresponds to the numerical solution of the trivial non-stiff ODE model $y' = 0$, obviously the special case of the Dahlquist test model for $\lambda = 0$; it assures that the obtained numerical solution dominates any possibly occurring "parasitic" numerical artifacts. For more details see, e.g., [16, Section 7.3.1]. This goes with the fact that Adams methods are only useful for *non-stiff* problems.

Discretization Error Estimate

Of course, to start such a scheme requires suitable starting values. Under certain (unrealistic) assumptions on such starting values, a rough examination of these schemes yields the discretization errors

$$y_1 - y(\tau) = \mathcal{O}(\tau^p), \quad p = \begin{cases} k, & \text{if } \beta_k = 0 \\ k+1, & \text{if } \beta_k \neq 0 \end{cases} \tag{2.45}$$

To start a scheme of order $\bar{p} \geq 2$, one must implement a *start-up procedure*

$$p = 1, 2, \ldots, \bar{p} - 1\ .$$

The initial step is always an explicit Euler step. For the start-up procedure, the above error behavior (2.45) changes from step to step, i.e. the order of discretization error increases from step to step until it reaches the order level strived for.

PECE Methods

For $\beta_k \neq 0$ the scheme (2.43) is *implicit* and the order p is $k+1$, i.e. one gains one order in comparison with the explicit version. However, one now has to solve a nonlinear system for the unknown y_1. Originally, Adams had suggested to use Newton's method, while Moulton later suggested to use a fixed point iteration. Today one realizes a variant named PECE method. In this approach, one starts with the associated explicit Adams method to obtain some *predictor* y_1^P. This value is then inserted into formula (2.43), wherein the value $f(y_1^P)$ is *evaluated* (E) to supply a *corrector* value y_1^C, which, in turn, requires the *evaluation* (E) of $f(y_1^C)$. This is the first step of a fixed point iteration, which would converge under some condition of the kind

$$L\tau \leq \text{const}\,. \tag{2.46}$$

This condition directly reminds one of the Lipschitz condition (2.8), which indicates that this extension of the explicit Adams method, too, can only be expected to be useful for *non-stiff* ODE problems.

For historical reasons, the *explicit* Adams methods are also called *Adams-Bashforth* methods (dating back to 1883), whereas the *implicit* Adams methods including the PECE methods are often named as *Adams-Moulton* methods.[1]

Order and Step-Size Control

The *adaptive* control of order and step size within any multistep method is much more difficult than in the one-step case. We do not want to go too much into details here, but refer interested readers to the textbook [16, Section 7.4]. For a potential user of Adams methods, only a few general remarks seem to be appropriate:

- The coefficients $\beta_0, \ldots, \beta_k$ introduced in (2.43) can be calculated off-line, if a *uniform* grid with constant step size τ is realized, see (2.44). If, however, some *step-size control* is realized, then this leads to a *non-uniform* grid, which, in turn, changes the Lagrange polynomials L_j in every new integration step; as a consequence, the coefficients need to be recomputed in the course of the numerical computation whenever the grid changes. For this reason, the Lagrange representation is replaced by some so-called *Nordsieck* implementation, which allows an easier handling of varying grids.
- If one wants to vary the *order*, then an alternative implementation based on Newton's divided differences is preferred.
- Due to the fact that order and step-size control require different implementations, Adams methods gain most of their efficiency on uniform grids and with constant order, i.e. when the ODE to be solved exhibits some quite regular dynamics. For the same reason, they are less reliable and robust in the efficient numerical solution of problems with rapidly changing dynamics.

Adams Codes

A rather efficient modern *adaptive* PECE code is DEABM (abbreviating Adams-Bashforth-Moulton) due to L. F. Shampine and H. A. Watts from 1979 or LSODE

[1]Moulton published his method not earlier than 1926, since it was regarded as a "military secret" during World War I.

(E stands for explicit) due to A. C. Hindmarsh [36] from 1980. All of these implementations have a natural way of realizing a "dense output" option, since they are based on interpolation.

2.3 Implicit Numerical Integration Methods

In this section, we present two types of efficient numerical *stiff integrators*, which are extensions of the *implicit* Euler method. Their implicit structure requires the numerical solution of *nonlinear* equations in each discretization step; this is realized via Newton-type methods, which require a linear equation solve in each iteration until convergence. It should be explicitly mentioned that solving these nonlinear equations by some fixed point iteration would not be appropriate, since this would assume some condition like (2.46), in contradiction to the intended treatment of stiff equations. First, in Sect. 2.3.1, we discuss *collocation methods*, the most efficient stiff integrators of implicit one-step or Runge-Kutta type. Next, in Sect. 2.3.2, we elaborate the *backward differentiation method* (in short: *BDF method*), the optimal candidate among multistep methods for stiff problems.

2.3.1 Collocation Methods

In Sect. 2.2.1, the general structure of explicit Runge-Kutta schemes has been shown. In 1963, J. C. Butcher [10] extended these schemes in an interesting way.

Implicit Runge-Kutta Schemes

According to Butcher, an *s-stage implicit Runge-Kutta scheme* is denoted by

$$k_i = f\left(y_0 + \tau \sum_{j=1}^{s} a_{ij} k_j\right), \quad i = 1, \ldots, s$$

$$y_1 = y_0 + \tau \sum_{i=1}^{s} b_i k_i$$

In general, such a scheme is no longer recursive in the unknown directions k_i, but requires the solution of a system of $s \cdot d$ *nonlinear* equations for $k_1, \ldots, k_s$. Any thus defined Runge-Kutta method can again be characterized by coefficients $(b, \mathcal{A})$, where b is an s-vector, but now $\mathcal{A}$ is a full (s, s)-matrix. In order to establish a general RK method of order p, one has to solve N_p algebraic equations (see Table 2.1) for the $s^2 + s$ coefficients, which gives a lot more degrees of freedom than in the merely explicit RK case. The extension to the *non-autonomous* case with $f(t, y)$ again uses

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, \ldots, s \;, \tag{2.47}$$

see (2.29) in Remark 5 for explicit RK methods.

L-Stability Conditions

For the class of general RK methods, there exists a simple necessary condition for L-stability (recall (2.28) above), which we briefly want to derive here. Insertion of the Dahlquist test model $y' = \lambda y, y(0) = 1$ into the above RK formulas yields (in matrix-vector notation and with $e^T = (1, \ldots, 1)$)

$$y_1(z) = 1 + zb^T(I - z\mathcal{A})^{-1}e = 1 + b^T(\frac{1}{z}I - \mathcal{A})^{-1}e \;.$$

Obviously, if the matrix $\mathcal{A}$ can be assumed to be nonsingular, then one ends up with the result

$$y_1(\infty) = 1 - b^T\mathcal{A}^{-1}e \;. \tag{2.48}$$

This formula has an interesting consequence. Assume the vector b is equivalent to any row (say j) of the matrix $\mathcal{A}$, i.e. $b^T = e_j^T\mathcal{A}$, then one gets

$$y_1(\infty) = 1 - e_j^T\mathcal{A}\mathcal{A}^{-1}e = 1 - e_j^Te = 0 \;. \tag{2.49}$$

In other words: If the vector b is equivalent to any row of the nonsingular matrix $\mathcal{A}$ and the corresponding implicit RK method is A-stable, then it is L-stable. We will make use of this property below.

Collocation Approach

Rather than constructing implicit RK methods via solving the many algebraic equations for the coefficients, one may construct a subset of such methods, called *collocation methods*, by some direct approach. Therein the continuous solution $y(t)$ is approximated by some polynomial function $u(t)$ with

$$u(0) = y_0, \quad u(\tau) = y_1$$

that satisfies the following *collocation* conditions:

$$u'(c_i\tau) = f(u(c_i\tau)), \quad i = 1, \ldots, s\ . \tag{2.50}$$

The prescribed (normalized) collocation nodes

$$0 \leq c_1 < \ldots < c_s \leq 1$$

characterize the method. Assuming such a polynomial exists, we may introduce a Lagrange basis $\{L_1, \ldots, L_s\}$ with respect to the nodes c_i. If we identify

$$k_i = u'(c_i\tau), \quad i = 1, \ldots, s\ ,$$

we may write the polynomial derivative as

$$u'(\theta\tau) = \sum_{j=1}^{s} k_j L_j(\theta)\ . \tag{2.51}$$

Upon integrating this relation, we obtain

$$u(c_i\tau) = y_0 + \tau \int_{\theta=0}^{c_i} u'(\theta\tau)d\theta = y_0 + \tau \sum_{j=1}^{s} a_{ij}k_j\ ,$$

where we have defined

$$a_{ij} = \int_{\theta=0}^{c_i} L_j(\theta)d\theta, \quad i,j = 1, \ldots, s\ . \tag{2.52}$$

Insertion of these definitions into the collocation conditions (2.50) verifies that collocation methods are in fact special implicit RK methods. Moreover, we obtain

$$y_1 = y_0 + \tau \int_{\theta=0}^{1} u'(\theta\tau)d\theta = y_0 + \tau \sum_{j=1}^{s} b_j k_j\ ,$$

where we have defined

$$b_j = \int_{\theta=0}^{1} L_j(\theta)d\theta, \quad j = 1, \ldots, s\ . \tag{2.53}$$

Thus we have all pieces of an implicit RK method together: With the choice of the (normalized) collocation nodes $c_1, \ldots, c_s$ we can define the Lagrange polynomials $L_1, \ldots, L_s$ and thus via (2.52) and (2.53) the coefficients $(b, \mathcal{A})$.

Discretization Error

The discrepancy between the ODE solution and the collocation polynomial is given by

$$y(\tau) - u(\tau) = \int_{\theta=0}^{1} f(y(\theta\tau))d\theta - \sum_{j=1}^{s} b_j f(u(c_j\tau)) = \mathcal{O}(\tau^{p+1}) \ . \tag{2.54}$$

Obviously, this represents some *quadrature* error (see, e.g., [17, Section 9.2]) of consistency order p.

In order to determine the maximally possible consistency order p, we arrive at the *Gauss-Legendre* quadrature, see, e.g., [17, Section 9.3]. In this algorithm, the collocation nodes are just the zeros of the *Legendre polynomial* $P_s(\theta), \theta \in [0, 1]$, which satisfy

$$0 < c_1 < \ldots < c_s < 1 \ .$$

The corresponding *Gauss collocation method* exhibits the following properties:

- Consistency order $p = 2s$ (maximum possible order).
- A-stability with $\mathcal{S} = \mathbb{C}_-$.
- Asymptotic behavior $|y_1(\infty)| = 1$.

This class of methods has further intriguing properties, which are discussed in [16, Section 6.3.4], but usually do not play a role in applications within systems biology. Here we would prefer an L-stable discretization method. That is why, in view of (2.49), a quadrature rule with $c_s = 1$ is selected, also named *Radau quadrature*. The remaining coefficients $c_1, \ldots, c_{s-1}$ are then the zeros of the *Jacobi polynomial* $P_{s-1}^{(0,1)}(\theta)$ so that

$$0 < c_1 < \ldots < c_{s-1} < 1 \ .$$

Under this sufficient assumption, the coefficient matrix $\mathcal{A}$ can be shown to be nonsingular (proof skipped here).

The thus constructed *Radau collocation method* is characterized by the following properties:

- consistency order $p = 2s - 1$,
- A-stability $\mathbb{C}_- \subset \mathcal{S}$,
- asymptotic stability $y_1(\infty) = 0$.

Hence, this collocation method is L-*stable*, as desired.

Radau Collocation Codes

A highly efficient code, named RADAU5, has been implemented by E. Hairer, see [33], and can be downloaded from his homepage. This code, obviously realizing the case $s = 3, p = 5$, uses a Newton-like iteration to solve the arising sd nonlinear equations. In each of these iterations, the direct solution of the linear equations is speeded up by factor of 5 exploiting the special structure of the arising matrix. Since the method is of collocation type, there exists a natural interpolation $u(\theta\tau)$ for all values $\theta \in [0, 1]$, which is the basis for a "dense output" option; in turn, this opens the door to a possible application to *delay* or *retarded differential equations*, see (1.5). The corresponding code RADAR5 due to N. Guglielmi and E. Hairer, see [31], is a direct extension of RADAU5 for this class of problems.

2.3.2 BDF Method

In this multistep approach, we start from the differential equation in its original form at time point $t_1 = \tau$, i.e.

$$y'(\tau) = f(y(\tau)) \tag{2.55}$$

and replace the unknown solution y by a polynomial g_k interpolating the $k + 1$ "previous" values

$$y_1, y_0, y_{-1}, \ldots, y_{-k+1}$$

on the uniform grid $t_1, t_0, t_{-1}, \ldots, t_{-k+1}$ with stepsize τ. This polynomial is uniquely defined so that we may write

$$y(t) \quad \to \quad g_k(t) = \sum_{j=0}^{k} L_j(t) y_{j-k+1}$$

where the L_j denote the Lagrange polynomials that depend on the interpolation nodes only, see, e.g., [17, Section 7.1]. Insertion of the expression

$$g_k'(\tau) = \sum_{j=0}^{k} L_j'(\tau) y_{j-k+1} = \frac{1}{\tau} \sum_{j=0}^{k} L_j'(0) y_{j-k+1}$$

into (2.55) yields the discretization scheme

$$\alpha_k y_1 + \ldots + \alpha_0 y_{-k+1} = \tau f(y_1) \tag{2.56}$$

in terms of uniquely defined coefficients $\alpha_j = L_j'(0)$. This special implicit multistep method has been suggested for stiff integration by C. W. Gear [28] in 1971. The relation (2.56) may be interpreted as an interpolation formula for numerical differentiation based on backward values, which gave the name *backward differentiation formula*, briefly: BDF.

In each step of a k-order BDF method, a system of only d nonlinear equations for y_1 has to be solved; note that the dimension of the system is independent of the order k. For $k = 1$ the implicit Euler method arises. The nonlinear systems must be solved by some Newton-like method, since any fixed point iteration would not be appropriate for stiff ODE problems. Consequently, several linear systems of the kind (ignoring the specific argument by merely writing $(\cdot)$)

$$\left(\alpha_k I_d - \tau f_y(\cdot)\right) \Delta y = \tau f(\cdot) \ .$$

must be solved until convergence of the Newton-like iteration.

Stability Properties

Upon inserting the usual Dahlquist test model (2.23), the discretization (2.56) yields

$$y_1(z) = -\frac{\alpha_{k-1} y_0 + \ldots + \alpha_0 y_{-k+1}}{\alpha_k - z} \ .$$

For $z \to \infty$ we thus obtain

$$y_1(\infty) = 0$$

which clearly looks like an extension of one of the conditions for L-stability of one-step methods. Of course, we would need A-stability in addition. However, stability analysis of multistep discretizations is much more subtle than for one-step methods; for the purpose of this book we do not dwell on the details. Important in our context is the so-called *second Dahlquist barrier* (see, e.g., [16, Section 7.2.2]): It states that *multistep methods with order $k > 2$ cannot be A-stable*, which rules out L-stability for $k > 2$. As the coefficients are determined, we must be content with L(α)-stability as it comes out, see Table 2.3. Not shown in the table is that the method is not even consistent for $k > 6$.

Due to the drastic deterioration of L(α)-stability for increasing order, *oscillatory phenomena* should be computed with order $k \leq 2$. There are well-known test examples with oscillatory behavior where BDF methods of higher order slow down

Table 2.3 L(α)-stability of BDF methods. Observe the second Dahlquist barrier

k	1	2	3	4	5	6
α	90°	90°	∼86°	∼73°	∼52°	∼18°

significantly or even produce wrong results, see [14]. Recall that $k = 1$ alone is not a really good idea, since the implicit Euler method is known to exhibit the unwanted "superstability".

Order and Step-Size Control

As for the Adams scheme, the BDF scheme, too, requires suitable starting values. Under certain (unrealistic) assumptions on such starting values, a rough examination supplies the discretization error

$$y_1 - y(\tau) = \mathcal{O}(\tau^{k+1}), \tag{2.57}$$

To start a scheme of order $\overline{k}$, one must implement a *start-up procedure*

$$k = 1, 2, \ldots, \overline{k} \leq 6 \ .$$

The starting step is always an implicit Euler step. With such a procedure, the above discretization error bound changes from step to step.

Just as in the Adams case, algorithmic difficulties in the *adaptive* selection of order and step sizes arise. The derivatives of the Lagrangian polynomials change whenever the local step sizes change, leading to some non-equidistant grid. This leads to the dilemma of implementation as either a Nordsieck variant or a divided difference variant. By and large, the method gains its best efficiency when run with constant step size and fixed order, which makes it a bit less robust than one-step stiff integrators.

BDF Codes

Among the most popular and efficient BDF codes are: the code LSODI (I stands for implicit) due to A. C. Hindmarsh [36]; the code LSODA (A stands for automatic switching) which automatically switches between the (implicit) BDF and the (explicit) Adams method; VODE due to [8], a variable step size/variable order BDF code, or the most recent code DASSL due to L. Petzold [51]. Due to their structure based on interpolation, a natural "dense output" option is usually available.

2.4 Linearly Implicit One-Step Methods

This section is devoted to efficient numerical *stiff integrators* that, in contrast to the methods of the previous Sect. 2.3, merely require the numerical solution of a low fixed number of *linear* equations per discretization step.

General Idea

The key issue in numerical stiff integration is the correct treatment of asymptotic stability of a given ODE model by step sizes that reflect the smoothness of the "slow" part of the solution, not that of the "fast" transition part of the solution, in cases where this is of less interest. In order to tackle this issue, one may subtract a linear homogeneous term on both sides of the ODE thus obtaining

$$\underbrace{y' - Jy}_{\text{implicit}} = \underbrace{f(y) - Jy}_{\text{explicit}} =: \bar{f}(y), \qquad y(0) = y_0 \, . \tag{2.58}$$

For stability reasons, the matrix J herein is either the Jacobian $J = f_y(y_0)$ or some approximation of it. The idea is to discretize the linear part on the left side implicitly (hence the name), but the "deflated" right-hand side $\bar{f}(y)$ explicitly. Clearly, such discretization schemes are computationally easier to realize than the fully implicit schemes from the preceding Sect. 2.3.

2.4.1 Rosenbrock-Wanner Methods

In 1963, H. H. Rosenbrock suggested a linearly implicit extension of explicit RK methods, which was later modified and improved by G. Wanner [33]. That is why such schemes today are called *Rosenbrock-Wanner* (ROW) schemes:

$$\Big(I - \tau\beta_{ii} J\Big) k_i = \tau \left[\sum_{j=1}^{i-1} (\beta_{ij} - \alpha_{ij}) \, J k_j + f\Big(y_0 + \tau \sum_{j=1}^{i-1} a_{ij} k_j \Big) \right] \tag{2.59}$$

The fact that the first right-hand sum above ends at index $i-1$ indicates that the system is block-triangular, while the second sum is anyway explicit. If the β_{ii} are all different, then a sequence of s linear systems with (d, d)-matrices $I - \tau\beta_{ii}J$ must be solved numerically. To simplify the linear algebra, very early the choice

$$\beta_{ii} = \beta, \; i = 1, \ldots, s$$

has been suggested, which implies that only one matrix $I - \tau\beta J$ needs to be decomposed throughout all stages.

In the above ROW methods, the identification $J = f_y(y_0)$ is strictly assumed. As in general RK methods, the coefficients $(\alpha_{ij}), (\beta_{ij})$ must be determined such that $N_p^{\text{ROW}} = N_p^{\text{RK}}$ algebraic equations corresponding to order p must be satisfied, see Table 2.4. If, however, the Jacobian J is replaced by an arbitrary Jacobian approximation matrix W, then one speaks of W-*methods*. As a consequence, a larger number N_p^{W} of algebraic equations needs to be satisfied, see again Table 2.4.

Table 2.4 Number of algebraic conditions to be satisfied by coefficients (α_{ij}), (β_{ij}) of ROW- versus W-methods

p	1	2	3	4	5	6	7
N_p^{ROW}	1	2	4	8	17	37	85
N_p^{W}	1	3	8	21	58	166	498

ROW Codes

As in Runge-Kutta methods, a whole "ROW-technology" has evolved over the years that led to a large number of implementations. In most cases, ROW methods of low order have been developed, for reasons clear from Table 2.4. Construction principles were, of course, the desirable L-stability and economy of evaluations as well as matrix decompositions and forward/backward substitutions. Among the most efficient codes of this kind are certainly

- ROS3PL with $p = 3$ and $s = 4$ due to J. Lang and D. Teleaga [43], an L-stable ROW-method, which is robust against Jacobian perturbations (even though not a full W-method),
- RODAS with $p = 4$ and $s = 6$ due to E. Hairer and G. Wanner [33]

All these codes are *adaptive*, which here means they possess an automatic step-size control, but keep the order fixed. A "dense output" option can be naturally realized within the embedding of the ROW methods.

2.4.2 Extrapolation Methods

As in the non-stiff case, the construction of a subset of linearly implicit Runge-Kutta methods can be directly realized via extrapolation, thus avoiding the cumbersome solution of the many algebraic equations for the coefficients. In such methods, one merely has to apply some well chosen basic discretization scheme of lowest order that is suitable for stiff integration. Higher orders are then obtained via the Aitken-Neville algorithm. Order and step-size control is realized just as in the explicit extrapolation methods. For ODE problems in systems biology, two kinds of schemes are useful.

Linearly Implicit Euler Discretization

Starting from the basic idea in (2.58), one discretizes the linear part on the left by the implicit Euler scheme, which leads to (for $n = 0, 1, \ldots$ and internal step size σ)

$$\eta_{n+1} = (I - \sigma J)^{-1} \left(\eta_n + \sigma \bar{f}(y_n)\right) = \eta_n + \sigma (I - \sigma J)^{-1} f(y_n) \ . \tag{2.60}$$

Table 2.5 Linearly implicit Euler scheme with extrapolation when subdivision $\mathcal{F}_H$ is chosen. L(α)-stability of subdiagonal elements $T_{k+1,k}$ in the extrapolation tableau

k	1	2	$3 \cdots 7$
α	90°	90°	≥89.77°

Note that this is formally some W-method, since the matrix J is not required to be the exact Jacobian $f_y(y_n)$, but will usually be selected as some Jacobian approximation $J \approx f_y(y_0)$. In passing we note that for $J = 0$ we obtain the extrapolation method based on the explicit Euler scheme. This basic scheme is run repeatedly with successively smaller internal step sizes (compare Sect. 2.2.2 above)

$$\sigma_k = \tau/n_k, \quad n_k \in \mathcal{F}_H = \{1, 2, 3, 4, \ldots\} .$$

This implies that n_k linear systems need to be solved requiring the decomposition of (d, d)-matrices $I - \sigma_k J$ and the corresponding number of forward/backward substitutions.

From theory, one knows that this discretization permits an asymptotic σ-expansion, see [16, Section 6.4.2]. This is the theoretical basis for some σ-extrapolation, see Table 2.2 for the corresponding triangular Aitken-Neville scheme to compute elements T_{ik}.

Stability properties. Insertion of the Dahlquist test model (2.23) confirms that all elements T_{ik} (with $i \geq k$) of the extrapolation tableau satisfy the necessary condition (let $z = \lambda\tau$ and $J = \lambda$)

$$T_{ik}(z) \sim \frac{1}{z^{i-k+1}} \to 0 \quad \text{for} \quad z \to \infty .$$

This is one of the necessary conditions for L-stability. In Table 2.5, we arrange the L(α)-results for the subdiagonal elements in the extrapolation tableau, which are actually chosen for error as well as order and step-size control.

Dense output. As usual for adaptive extrapolation integrators, this one also selects rather large step sizes due to its efficient order and step-size control. A cubic Hermite interpolation tool has been constructed in [19] based on the information y_0,$f(y_0)$ and y_1,$f(y_1)$. In the DAE case, good approximations for $f(y_1)$ are obtained via σ-extrapolation based on the values $(\eta_n - \eta_{n-1})/\sigma$ for $n \in \mathcal{F}_H$. An efficient higher order technique has been worked out by E. Hairer and A. Ostermann [32].

Linearly Implicit Midpoint Rule

As in the non-stiff case, we would prefer to construct some basic discretization scheme that permits σ^2-extrapolation. This has been achieved by G. Bader and P. Deuflhard [3]. As a symmetric extension of the explicit mid-point rule, they introduced the *linearly implicit midpoint rule* according to

$$(I - \sigma J)\eta_{n+1} - (I + \sigma J)\eta_{n-1} = 2\sigma\bar{f}(y_n) \tag{2.61}$$

to be started by a linearly implicit Euler step. Instead of an extension of the Gragg final step (2.38), they introduced a different symmetric final step, which also requires an additional evaluation of $f(\eta_{n_\nu})$,

$$y_1(\tau;\sigma_\nu) := \hat{\eta}_{n_\nu} = \frac{1}{2}(\eta_{n_\nu+1} + \eta_{n_\nu-1}), \quad n_\nu \text{ even .} \tag{2.62}$$

The reason for this final step will be explained below in the context of stability.

Stability properties. Insertion of the Dahlquist test model (2.23) yields

- for the odd indices

$$\eta_{2m+1} = \frac{1}{1-z}\left(\frac{1+z}{1-z}\right)^{m-1} \to \frac{(-1)^m}{z} \to 0\ , \tag{2.63}$$

- for the even indices

$$\eta_{2m} = \left(\frac{1+z}{1-z}\right)^{m} \to (-1)^m \to 0\ , \tag{2.64}$$

- and for Bader's symmetric final step

$$\hat{\eta}_{2m} = \frac{1}{2}(\eta_{2m+1} + \eta_{2m-1}) \to \frac{(-1)^{m-1}}{z^2} \to 0\ ,$$

  which can be seen to perform some asymptotic smoothing.

In order to have the same asymptotic sign pattern for all smoothing steps, one arrives at the sequence $\mathcal{F}_\alpha$ shown below. Clearly, with these specifications, the asymptotic result

$$T_{ik}(z) \sim \frac{-1}{z^2} \to 0 \quad \text{for} \quad z \to \infty$$

is obtained, i.e. one has a uniform asymptotic pattern throughout the whole extrapolation table. We are thus only left to study the L(α)-stability pattern, which is given in Table 2.6.

Table 2.6 Linearly implicit mid-point rule with extrapolation when subdivision sequence $\mathcal{F}_\alpha$ is chosen. L(α)-stability of subdiagonal elements $T_{k+1,k}$ in the extrapolation table

k	1	2	3	4	5	6
p	1	3	5	7	9	11
α	90°	90°	∼88°	∼86°	∼87°	∼87°

Obviously, the stability properties including the final step are very satisfactory. However, from the detailed study of the intermediate discretization steps we learned that the asymptotic behavior (2.63) for the odd indices is satisfactory, while the behavior (2.64) for the even indices is unsatisfactory. As a consequence, this extrapolation method is recommendable mainly for "moderately stiff" problems, which, however, represent the typical problems in systems biology.

Quadratic extrapolation. For a scheme as specified, the existence of an asymptotic σ^2-expansion has been shown in [3]; it gives rise to a σ^2-extrapolation for *even* subdivision sequences. Additional conditions come from the above stability analysis that lead to the sequence

$$\sigma_k = \tau/n_k, \quad n_k \in \mathcal{F}_\alpha = \{2, 6, 10, 14, 22, 34, 50\}\,.$$

The index α comes from the empirically introduced property $n_{k+1}/n_k \le \alpha := 1.4$. These data enter into a triangular Aitken-Neville scheme with T_{ik} for $i \ge k$ as shown in Table 2.2. Needless to say that, of course, an adaptive version is implemented, see (2.41) for the step size control and (2.42) for the order control, the latter requiring some subtle decision about what should be inserted as computational work per discretization step.

Dense output. As in the other extrapolation integrators, a cubic Hermite interpolation polynomial can be constructed based on the information $y_0, f(y_0)$ and $y_1, f(y_1)$. The idea is the same as for the explicit midpoint rule: at the end of each τ-step one evaluates $f(y_1)$, which then can be recycled as $f(y_0)$ in the next integration step so that no additional function evaluation is needed (apart from the very last step). Again we mention that the accuracy of this Hermite interpolation formula in combination with the order and step-size control is enough for the purpose of systems biology.

Linearly Implicit Extrapolation Codes

The *adaptive* extrapolation code EULSIM has been designed on the basis of the linearly implicit Euler scheme (originally called S*emi-* IM*plicit* Euler scheme, hence the name). The more elaborate extrapolation code LIMEX due to [19, 23] is an extension that also applies to quasilinear differential-algebraic equations. For "moderately stiff" ODE problems, which in systems biology are the most frequent case, the code METAN1 [3], based on the linearly implicit midpoint rule, typically supersedes LIMEX. For an illustration, see Sect. 2.5.2. In both codes, dense output options are available based on cubic Hermite interpolation. The code METAN1 also found its way into the book on "Numerical Recipes" [52], p. 735.

2.5 Choice of Numerical Integrator

A code may fail; but it must not lie.
(Beresford N. Parlett)

In the sections above, a number of different methods including efficient codes have been discussed to necessary detail. The present section is devoted to questions that a user may have when deciding which of these codes to apply for his problem at hand. In Sect. 2.5.1, we arrange methods and associated codes (written as CODES) again; for download addresses see the final chapter *Software* at the end of the book. In Sect. 2.5.2, we illustrate the choice of integrator at two moderate size examples from systems biology, which look rather similar in terms of the differential equation model, but behave differently in terms of the numerics. Finally, in Sect. 2.5.3, we present a quite challenging large scale problem dealing with the so-called Warburg effect of tumor cells.

2.5.1 A General Roadmap for Numerical Integrators

One-Step Methods

In the sections above we have discussed *explicit* one-step methods for *non-stiff* ODE problems such as

- explicit Runge-Kutta methods in Sect. 2.2.1 (DOPRI5 , DOP853), and
- extrapolation methods in Sect. 2.2.2 (DIFEX1, ODEX) ,

as well as *(linearly) implicit* methods for *stiff* problems such as

- Radau collocation methods in Sect. 2.3.1 (RADAU5, RADAR5),
- Rosenbrock-Wanner methods in Sect. 2.4.1 (ROS3PL, RODAS), and
- extrapolation methods in Sect. 2.4.2 (LIMEX, METAN1).

All of these codes select a "locally optimal" step size on the basis of local discretization error estimates, extrapolation methods also a "locally optimal" order, codes like DOP853 an optimal order among the three orders 8,5,3. In parallel with the dynamics of the ODE system, *non-uniform* grids are obtained with *problem dependent* output points, typically much less than with uniform grids. If more than the automatically computed output data are wanted, which is often called the "dense" output option, then extra tools for interpolation are appropriate to avoid wasting computing time due to "too many" output points; such extra tools are easily available in embedded RK methods as well as in collocation methods, while extrapolation methods require an additional (computationally cheap) device based on Hermite interpolation. Compared to multistep methods, implementations of these methods exhibit only a small amount of overhead beyond f-evaluations. Generally speaking, these methods are particularly efficient in ODE problems with strongly varying dynamics, a feature that is especially true for extrapolation methods and DOP853, since they additionally choose some "locally optimal" order.

Multistep Methods

From the class of multistep methods we have discussed

- (*explicit*) Adams methods for *non-stiff* ODE problems in Sect. 2.2.3 (LSODE, DEABM),
- (*implicit*) BDF methods for *stiff* problems in Sect. 2.3.2 (LSODI, VODE, and DASSL), and
- a multistep code that *automatically switches* between Adams and BDF method (LSODA).

All of these codes realize some control of order and step size, but in a much more restricted sense than in one-step methods. Efficient implementations for a change of *order* are different from those for a change of *step sizes*. Generally speaking, multistep methods gain their efficiency with *quasi-uniform* grids, a property prohibitive for problems with strongly varying dynamics, but in favor of smoothly varying dynamics; this goes with the intuition that a smooth dynamics can gain efficiency from exploiting the "history" of the trajectory. For smooth dynamics, the number of evaluations of the right-hand side f may be considerably less compared to one-step methods. For the BDF method, an order restriction $k \leq 2$ is recommended when applied to oscillatory problems, which do occur in systems biological networks. By their common construction principle via interpolation, both Adams and BDF methods naturally generate "dense" output, see Sect. 2.2.1. In comparison with one-step methods, they usually require much more computational overhead which often outweighs the possibly lower number of f-evaluations.

Non-stiff Versus Stiff Integration

The question of whether a given ODE problem should be regarded as stiff or non-stiff, stands at the beginning of each systems biological simulation. For a non-stiff problem, an explicit method will do, which only requires evaluations of the right-hand sides f and thus is faster per integration step than an implicit method. For a stiff problem, implicit or a linearly implicit methods may pay off, which additionally require the numerical solution of linear equations involving the Jacobian of the right-hand side, but over significantly less integration points. For really stiff problems, an

explicit method would suffer from severe *step-size restrictions* that would blow up the overall computing time.
Qualitative insight. As for the practical classification "stiff/non-stiff" in a given ODE problem, quite often some qualitative insight is helpful before starting the simulation: stiff problems are typically characterized by the fact that they asymptotically approach some *steady state point*. Then, generally speaking, an implicit or linearly implicit method should be applied. However, even for such problems, an explicit method might be preferable, if one is only interested in the "transition" phase.
Rule of thumb. As for the theoretical distinction between stiff and non-stiff ODEs, we have elaborated quite a bit on this question, but finally recommended a rather pragmatic approach: ODE problems, wherein the extra computational amount required for the arising nonlinear (or linear) equations pays off, are regarded as stiff. Consequently, whenever a stiff/non-stiff characterization of an ODE problem is unclear, the following rule of thumb is advised: start with an explicit method, say DOPRI5, and only switch to an implicit or linearly implicit method, say METAN1, if the explicit method seems to suffer from step-size restrictions that seem "uninterpretable" in view of the underlying model.

For an illustration of stiff versus non-stiff ODE problems, the following example may serve.

Example 8 (Gene expression) Let a gene expression be described by the scheme

$$\text{gene} \xrightarrow{k_1} \text{mRNA} \xrightarrow{d_1} \emptyset,$$

$$\emptyset \xrightarrow{k_2 \cdot [\text{mRNA}]} \text{protein} \xrightarrow{d_2} \emptyset,$$

wherein k_1 is the constitutive transcription rate, k_2 the translation rate, d_1 the mRNA degradation rate, and d_2 the protein degradation rate. We assume that this gene expression is unregulated, i.e. the gene is always on, which can be modelled by setting its concentration $g = [\text{gene}] \equiv 1$. Let $m = [\text{mRNA}]$ the concentration of mRNA and $p = [\text{protein}]$ the concentration of the protein. Thus one arrives at the ODE initial value problem

$$m' = k_1 - d_1 m, \; m(0) = 1, \quad p' = k_2 m - d_2 p, \; p(0) = 0 \; .$$

Let the kinetic parameters be selected as $k_1 = 2$, $d_1 = 1$, $k_2 = 1$, $d_2 = 0.01$. The steady state point is $m^* = k_1/d_1$, $p^* = (k_1 k_2)/(d_1 d_2)$. The numerical solution is shown in Fig. 2.10. As can be observed, mRNA reaches its steady state much faster than the protein. This implies that the problem can be regarded as stiff. In Fig. 2.11, the step sizes chosen by a non-stiff integrator (here: DOPRI5) are compared with those chosen by a stiff integrator (here: LIMEX).

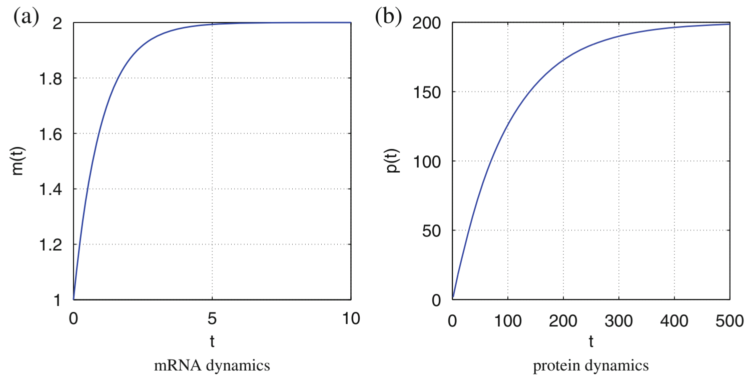


Fig. 2.10 Numerical solution for Example 8. Note the different time-scales in the two plots: The mRNA concentrations changes much more rapidly than the protein concentration

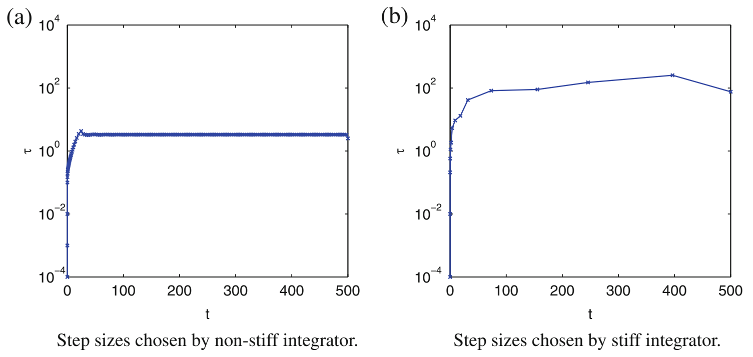


Fig. 2.11 Example 8. Automatic step-size selection by two different integrators. Common local error tolerance TOL= 10^{-6}. *Left*: Non-stiff integrator DOPRI5. "Unreasonable" step-size restriction with many ups and downs in the steady state phase. *Right*: Stiff integrator LIMEX. No step-size restriction in the steady state phase

Remark 8 For quite a while research has focused on the construction of methods that *automatically* classify stiff versus non-stiff problems, switching between explicit and implicit codes during the computation. An example of this type is the quite popular multistep code LSODA that automatically switches between stiff (BDF) and non-stiff (Adams) methods. There is, however, a principal difficulty: The distinction is easy in an implicit method (here: BDF), since there Jacobian information is available; but then, in a non-stiff problem, the bulk of the computational amount has already been spent so that not too much computing time can be saved. The distinction is difficult in an explicit method (here: Adams), since there the necessary information on the Jacobian matrix is missing.

Computational Speed

Suppose the decision between non-stiff or stiff integrator has been made. Then the choice among explicit or implicit integrators, respectively, must be made. The task to find the "fastest" code for the problem at hand is not as easy as one might expect. In published comparisons of computing times for different codes, the multistep community quite often only gives the number of f-evaluations excluding overhead, while the one-step community tends to present only total computing times, which often depend on the selected computer.

Non-stiff integrators. For this class of methods, the total computing time (CPU time) originates from the number N_f of f-evaluations (at a cost of C_f each) and the overhead $\Omega = \omega_{\text{method}} \cdot d$, usually proportional to the dimension d of the ODE system with a method dependent proportionality factor ω_{method}. Thus we arrive at

$$\text{CPU} = C_f \cdot N_f + \Omega = C_f \cdot N_f + \omega_{\text{method}} \cdot d \; .$$

As explained above, multistep (ms) and one-step (os) methods may be characterized by the relations

$$N_f^{\text{ms}} \leq N_f^{\text{os}}, \quad \omega_{\text{os}} \ll \omega_{\text{ms}} \; .$$

Obviously, the distinguishing quantity will be

$$\gamma = C_f / d \; ,$$

which can be interpreted as "evaluation time per component of the right-hand side f". From it, we may derive the following *rule of thumb*:

- Whenever γ is "not too large", then some explicit Runge-Kutta or extrapolation method should be chosen; in systems biology, this seems to be the most frequently occurring case, since each component typically couples only with few other components.
- If γ is "large", then one should prefer an Adams method.

If the expected dynamics is "strongly varying", then a one-step method should be taken anyway. Compared to the optimized explicit Runge-Kutta codes DOPRI5 and DOP853, the two extrapolation codes DIFEX1 or ODEX typically are regarded as slightly slower in standard non-stiff ODE problems, but slightly more robust in challenging real life problems.

Stiff integrators. For implicit and linearly implicit integrators there is no such *simple* complexity theory as in the non-stiff case. Here we have to additionally count the number of matrix decompositions and of forward/backward substitutions. This confuses the picture quite a bit. What remains valid is that the BDF method as a multistep method also requires a much larger overhead than the one-step competitors. Moreover, the linearly implicit one-step methods (Rosenbrock-Wanner or extrapolation method) are much simpler than their implicit counterparts, which particularly pays off for large ODE systems, when the arising linear systems may even be solved iteratively.

Accuracy

Recall from Sects. 2.1.1 and 2.1.2 that a user of a numerical integrator can only prescribe a local error tolerance TOL, typically split into RTOL, a relative error tolerance, and ATOL, an absolute error tolerance. Note that RTOL relies on an efficient scaling (see the associated item below). The achieved global accuracy ERR then depends on the condition of the problem. In real life problems, the condition number is usually computationally unavailable. Hence, a user should develop some "feeling" about the necessary error tolerance. As for the choice of the integrator, the achieved accuracy will mostly be better, if less integration points are needed – a feature that usually speaks for extrapolation methods.

Computational Parameter Sensitivity Analysis

In the class of problems envisioned here, the additional numerical integration of the sensitivity equations will usually come up. From Sect. 1.3.2, we recall that the sensitivity y_p with respect to some parameter p (dropping the index) is described by the equations

$$y_p' = f_y(y(t),p)y_p + f_p, \quad y_p(0) = 0 \ .$$

As already mentioned in Remark 4, these equations must be solved simultaneously with the original model equations $y' = f(y)$ to obtain the argument inside $f_y(\cdot), f_p(\cdot)$. The most convenient way to realize the above variational equation is to generate the exact formulas for f_y and f_p from some chemical compiler simultaneously with the generation of the right hand side f (see Sect. 1.2.3).

If some *stiff* integrator is employed, then the Jacobian f_y and the decomposition of the matrix $I - \beta\tau f_y(y_0)$ can be also included in the integration of the state variable ODEs. Within any *linearly* implicit one-step method, the same idea as in (2.58) works again in the form

$$\underbrace{y_p' - Jy_p}_{\text{implicit}} = \underbrace{(f_y(y,p) - J)y_p + f_p}_{\text{explicit}} \qquad y_p(0) = 0 \ . \tag{2.65}$$

An efficient implementation of this idea within the code LIMEX has been suggested and worked out by M. Schlegel et al. [55], which pays off especially in large ODE networks. For BDF methods, which require the iterative solution of *nonlinear* equations, an especially adapted technique has been worked out by T. Maly and L. Petzold [44].

Discontinuity Treatment

In some applications, the right-hand sides f contain discontinuities at certain points. In systems biology, such a situation typically occurs when different models are used to describe different processes before or after some characteristic event (day-night, say). If one ignores such points by just “overriding” them with a numerical integrator, then usually the accuracy after these points will be poor. This phenomenon occurs less marked, if the employed integrator is equipped with an automatic order control that allows for sudden local drops of order (as in extrapolation methods). In principle, however, this kind of difficulty should be tackled differently: The integration should be terminated at these points and restarted thereafter; in this way, the accuracy can be preserved. Note that this procedure is a structural disadvantage for any multistep method, which requires a restart from order $k = 1$ up to the locally optimal order after the discontinuity point. For this reason, certain one-step procedures have been designed to realize some “quick start-up”.

Example 9 For the purpose of illustration, we consider an artificial example constructed by R.D. Russell and L.F. Shampine [54]:

$$y'' = y - ty' + te^t - |t|(6 - 12t + 2t^2 - 3t^3),$$

$$y(-1) = e^{-1} - 2, \quad y'(-1) = e^{-1} + 7.$$

Here a discontinuity of y''' occurs at $t = 0$, known in advance. The unique solution is

$$y(t) = \begin{cases} e^t + t^3 - t^4, & t \le 0 \\ e^t - t^3 + t^4, & t \ge 0 \end{cases}.$$

After the point $t = 0$, accuracy will be reduced, if a numerical integrator without order control is used, see Fig. 2.12. On the contrary, stop and restart of the integration at $t = 0$ preserves the accuracy also beyond $t = 0$.

Dense Output

Such an option treats the case that more output points are wanted than automatically delivered by the step-size (and possibly order) control. This situation may well occur in systems biological modelling, there mainly for print-out. In addition, as worked out in the next Chap. 3, this option may also be important for parameter identification, when measurements are “too dense” compared with the step sizes selected by the adaptive integrators. Suppose one stopped the integrator at each of these points, then “too much” computational effort would be wasted. In particular, integrators with adaptive order control would find the lowest possible orders as optimal, if the distance between two neighboring points were “too small”. A dense output option

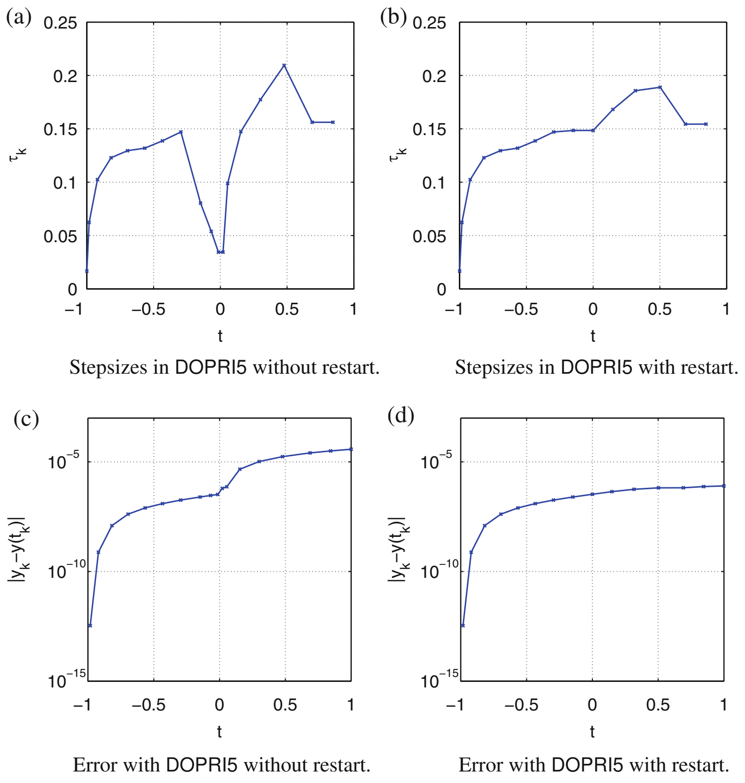


Fig. 2.12 Example 9. *Left:* A discontinuity of y''' at $t = 0$ causes step size reduction and accuracy loss. *Right:* Stop and restart of the integration at $t = 0$ preserves the accuracy also beyond $t = 0$

will usually leave the step-size control untouched and apply some interpolation formula for the points in between the automatically selected points. Such an option is easily implemented in multistep methods, Adams (Sect. 2.2.3) as well as BDF (Sect. 2.3.2) methods, since they are both constructed via interpolation formulas. In explicit RK methods, this option is conveniently realized via embedding techniques (Sect. 2.2.1). In the implicit Radau methods, there is anyway a natural representation of the whole trajectory via collocation. Extrapolation methods need to implement an extra device based on cubic Hermite interpolation; higher order dense output formulas have been worked out by E. Hairer and A. Ostermann [32].

Delay Differential Equations

As already mentioned in (1.5) in the introduction, the standard ODE system is sometimes replaced by a delay system. In biological systems, the *retardation* or *delay time* $\tau > 0$ typically depends nonlinearly on the solution y, so that instead of (1.5) one should better write

$$y' = f\left(y(t), y(t - \tau(y(t)))\right), \quad y(t) = \Theta(t) \text{ for } [-\tau, 0]$$

with a given *initial function* Θ. Numerical integrators with a dense output option also permit the treatment of *delay* or *retarded differential equations* (DDEs), see (1.5). Among the explicit RK methods, we mention the code RETARD due to E. Hairer, among the implicit RK methods the code RADAR5 due to N. Guglielmi and E. Hairer [31]. In MATLAB, the solver dde23 due to L. F. Shampine [56] is provided for DDEs with *constant* delays, a case rare in systems biology; this code tracks discontinuities and integrates numerically by the explicit Runge-Kutta (2,3) pair and an interpolant implemented within MATLAB's ode23.

Reliability

A numerical integrator is said to be "reliable", if the delivered solutions are "accurate enough" compared with the condition of the problem at hand. For a basic understanding of this issue recall Sects. 2.1.1 and 2.1.2 where the relation of local and global accuracy has been discussed in some detail. From this discussion we know that *local* discretization errors need to be estimated; these estimates then enter into some adaptive control of step sizes (and possibly also orders). The finally achieved accuracy additionally depends on the structure of the underlying ODE system and the number of integration points. In this respect, one-step methods, in particular extrapolation methods, have a natural advantage, since they require less integration points than multistep methods.

Robustness

In the world of mathematical ODE modelling, this is the most important property: a numerical integrator should solve a large class of given problems (no matter how difficult) without much ado. Apart from the many details discussed above, robustness typically requires additional heuristic strategies (e.g., avoid division by zero, find a reasonable starting step size) and a careful implementation of software.

Scaling

Robustness typically requires a subtle application of scaling techniques within the code, an intricate issue that we have not touched upon in this book. Apart from any *external* scaling of variables prescribed by the user, robust codes often realize some *internal* scaling. Such a device is necessary to assure that any relative versus absolute error criterion works. Moreover, quite often "small" elements of the numerical sensitivity matrices are set to zero – a device only reasonable when the term “small” is defined, which, in turn, is only reasonable, if scaled quantities are treated.

2.5.2 *Different Numerical Behavior in Two Similar Problems*

In this section, we present two problems from systems biology that, at first glance, look similar from the point of view of mathematical modelling. Both of them are of moderate size. They may serve as typical examples for how to deal with larger problems.

Human Menstrual Cycle Problem GynCycle

This model, published in detail in [53], has already been presented above as Example 1. In Fig. 1.6, the *compartments* of the model have been illustrated. In Fig. 1.7, part of the corresponding *chemical model* has been presented in the usual form of a reaction diagram. Thus one arrives at a *mathematical model* with $d = 33$ ODEs and 114 parameters, from which 63 degrees of freedom could be identified by methods described in the subsequent Chap. 3.

Bovine Estrous Cycle Problem BovCycle

This model has been inspired by the above human menstrual cycle model. In fact, the endocrine mechanisms that regulate the bovine estrous cycle are rather similar to those of the human menstrual cycle. A first version has been published in [7]. Our subsequently presented computations refer to the more recent version [58], where further details can be found. As for the selected *compartments* for the physiological description, Fig. 1.6 can again serve as defining the terms. A subdiagram of the *chemical model* is given in Fig. 2.13, to be compared with the more elaborate human model in Fig. 1.7. Finally, a mathematical model with $d = 15$ ODEs and 60 parameters comes up.

Comparative Performance of Numerical Integrators

In Figs. 2.14 and 2.15, we show the comparative performance of several numerical integrators. The documented local error tolerances TOL, see (2.14), range within $10^{-3}, \ldots, 10^{-6}$, which is a reasonable range for typical problems from systems biology. As for the performance, we study both the comparative CPU times required on a Fujitsu Siemens Lifebook E8210 and the achieved global accuracies ERR, see (2.15). The CPU time is a reasonable performance measure, whenever both

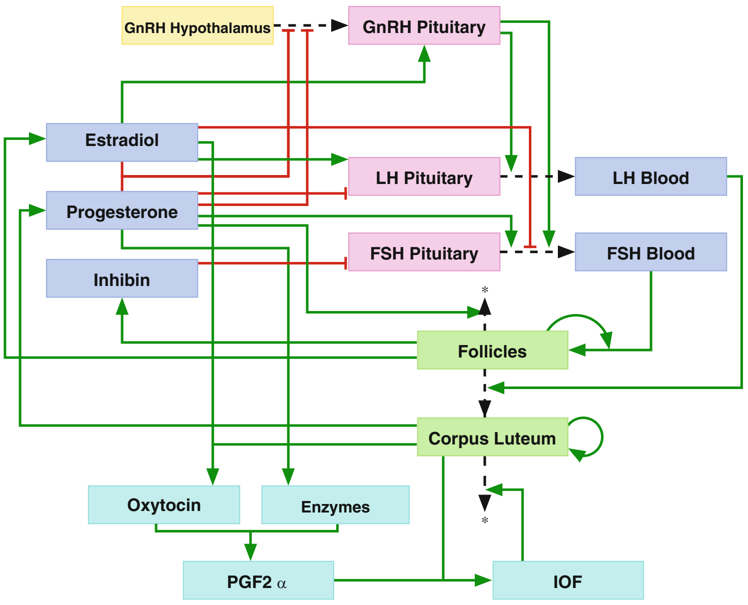


Fig. 2.13 Flowchart of a model for the bovine estrous cycle. *dashed lines*: transitions, elimination, or chemical reactions, *solid lines with filled arrows*: stimulatory effects (Hill functions h^+), *solid lines with unfilled arrows*: inhibitory mechanisms (Hill functions h^-)

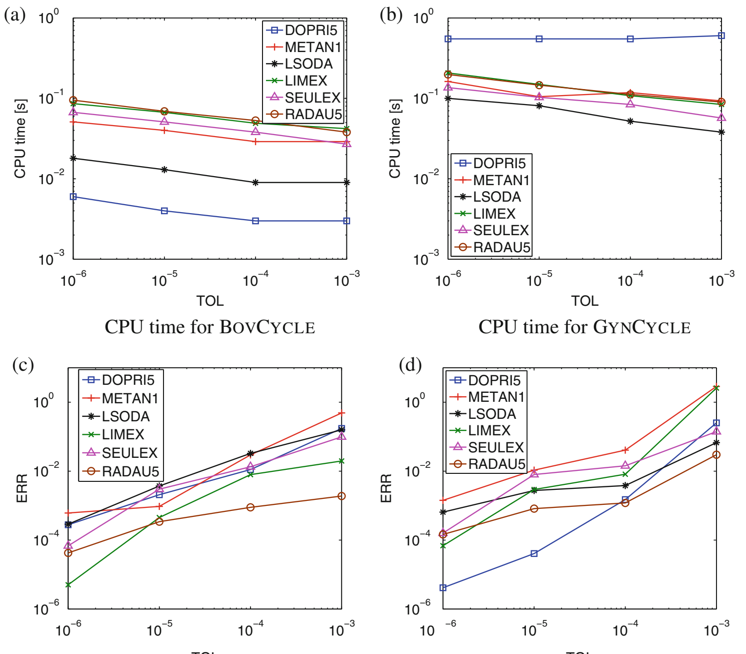
non-stiff and stiff integrators are compared. As for the norm used to define ERR, we selected the *scaled root mean square error*, i.e.

$$\mathrm{ERR} = \left(\frac{1}{d} \sum_{i=1}^{d} \frac{(y_i(T) - y_{i,\mathrm{ref}}(T))^2}{y_{i,\mathrm{scal}}(T)} \right)^{1/2} , \tag{2.66}$$

wherein y_{ref} is the (highly accurate) computational result obtained with the extrapolation code LIMEX for TOL = 10^{-12} and y_{scal} is the scaling vector obtained during the computation.

Small test set of integrators. First, in order come to a fast decision about which numerical integrator to use, we test on a small subset of integrators that includes both stiff and non-stiff ones. On the basis of what has been presented in this chapter, let us select the non-stiff Runge-Kutta integrator DOPRI5, the stiff extrapolation integrator METAN1, and the mixed multistep code LSODA with automatic switching between a non-stiff Adams method and a stiff BDF method. The comparative results are presented in Fig. 2.15. From these numbers, we may gain the following insight:

- Problem BovCycle is non-stiff, as can be seen from the small amount of computing time of DOPRI5 versus METAN1.



(a) CPU time for BovCycle (b) CPU time for GynCycle

(c) scaled root mean square error for BovCycle (d) scaled root mean square error for GynCycle

Fig. 2.14 Performance of six numerical integrators in two similar problems, to be compared with Fig. 2.15. *Left*: Bovine estrous cycle problem BovCycle. *Right*: Human menstrual cycle problem GynCycle. *Top*: CPU time. *Bottom*: Achieved global accuracy

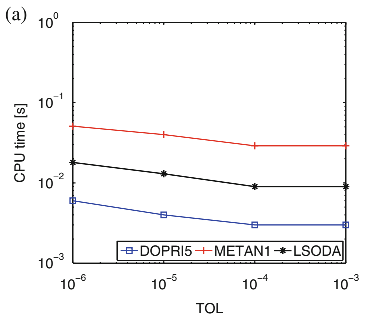
- Vice versa, problem GynCycle is stiff, just compare the higher CPU of DOPRI5 versus METAN1.
- In the non-stiff problem, the accuracies of all three integrators are nearly the same.
- In the stiff problem, the accuracies spread by a factor of roughly 10, with the non-stiff integrator DOPRI5 surprisingly best, which yields to the insight that this problem is only *mildly* stiff.

The *number of steps* selected by the automatic step-size controls (not presented in detail here) is

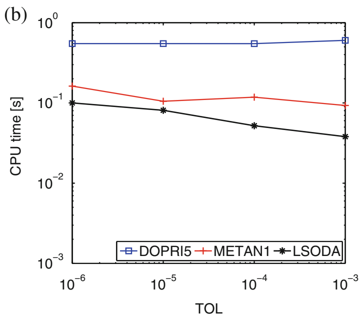
- for BovCycle: between 10^3 with LSODA, DOPRI5 and 10^2 with METAN1,
- for GynCycle: between 10^4 with DOPRI5 and again 10^2 with METAN1.

In summary, Fig. 2.15 leads to the suggestion of using DOPRI5 in the non-stiff problem BovCycle, but LSODA or METAN1 in the stiff problem GynCycle.

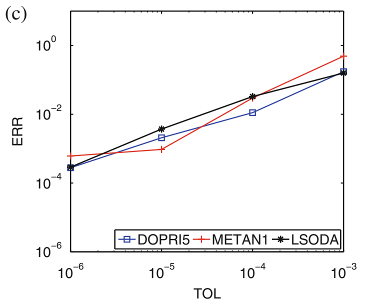
Larger test set of integrators. Sometimes, users want a comparison over a larger test set of integrators. Therefore, beyond the three integrators DOPRI5,



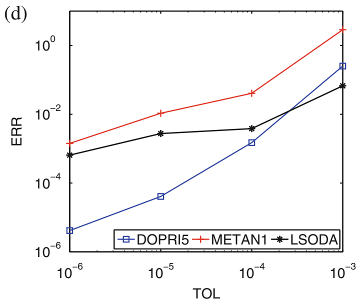
CPU time for BovCycle



CPU time for GynCycle



scaled root mean square error for BovCycle



scaled root mean square error for GynCycle

Fig. 2.15 Comparative performance of three numerical integrators in two similar problems. *Left*: Bovine estrous cycle problem BovCycle. *Right*: Human menstrual cycle problem GynCycle. *Top*: CPU time. *Bottom*: Achieved global accuracy

METAN1, LSODA above, we additionally include the three stiff integrators LIMEX, SEULEX, RADAU5. The comparative results are given in Fig. 2.14. From this larger data set, we conclude that the characterization "stiff versus non-stiff" remains the same. Moreover, the data about the computing times and achieved accuracies also remain essentially the same.

Remark 9 If, beyond the mere trajectory simulation, *sensitivity analysis* is wanted, then the linearly implicit extrapolation codes LIMEX, METAN1 have a structural advantage that also pays off as a gain in CPU time.

2.5.3 Example: Warburg Effect in Tumor Cells

This rather complex systems biological network has been worked out by M. König, H.-G. Holzhütter, and N. Berndt [42] from Charité, Berlin. We here partially follow their presentation. However, we focus on details of their numerical modeling, which is the topic of this monograph. Readers interested in more biological details are

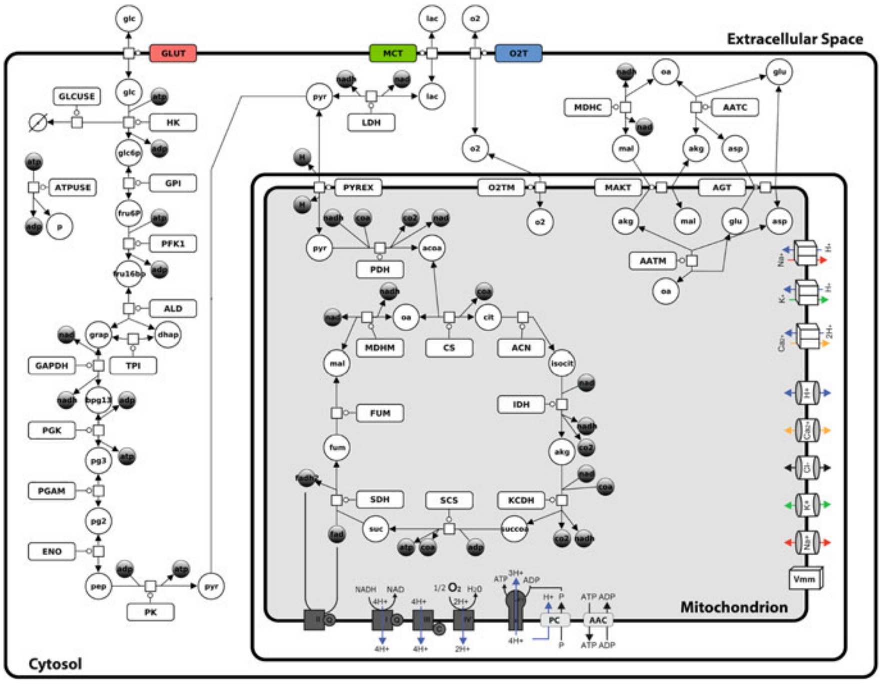


Fig. 2.16 Schematic representation of cellular-scale model for energy metabolism (Courtesy M. König)

referred to the original paper. The so-called *Warburg effect* means that solid tumor cells, as opposed to normal cells, exhibit an extraordinarily high demand for glucose even under aerobic conditions (i.e. in the presence of oxygen) with a substantial part of glucose being converted into lactate. In their work, the authors address the problem of whether zonation of the energy metabolism within a non-vascular tumor could serve as a means to influence its growth capacity.

Two-Scale Modeling

The authors of [42] developed an elaborate model for inter- as well as intra-cellular energy metabolism. Each cell is modeled via three compartments, the mitochondrion, the cytosol, and the extracellular space. Within these compartments, a kinetic model for 78 intracellular metabolites is realized. Figure 2.16 gives an impression of the complex metabolism within a single tumor cell. The most important processes are the tricarboxylic acid (TCA) cycle and the two central ATP delivering pathways, i.e. the glycolytic (GLY) pathway and the oxidative phosphorylation (OXP) pathway.

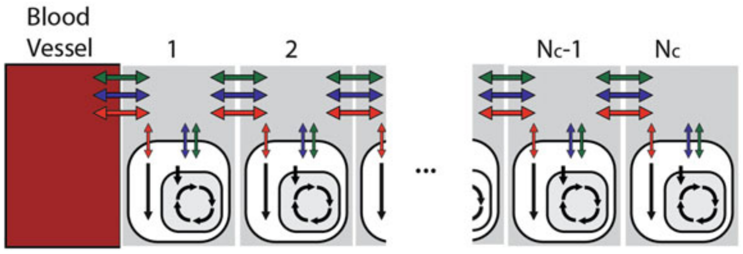


Fig. 2.17 Schematic representation of tissue-scale model of tumor metabolism. Note that Fig. 2.16 is a zoom into Fig. 2.17 (Courtesy M. König)

The coupling between tumor cells and with their tissue environment is again schematically described by a *compartment model*, see Fig. 2.17. The tumor as a whole is supplied with nutrients and oxygen via the nearest blood vessel (on the left of the figure). Exchangeable metabolites between the cells and the extracellular space are glucose, lactate and oxygen, which are assumed to diffuse between adjacent spatial layers – represented by double arrows in Fig. 2.17. In order to model the continuous diffusion process within the discrete compartment setting, a second-order finite difference approximation is used to replace the one-dimensional diffusion equation (which would be a partial differential equation).

Numerical Simulation

The total model above was specified to contain a fixed number of 25 cells, 5 extracellular compartments per cell, 78 intracellular metabolites per cell, and 3 extracellular metabolites. These model equations have been carefully programmed by hand to yield a system of 2328 ODEs. The project required repeated simulation runs within a *parameter study* that involved external oxygen and glucose availability or different metabolic strategies of energy production. This study required about 1000 simulation runs of the 2328 ODEs. Hence, computing time really matters here.

After first experiences of the authors with non-stiff integration, it quickly became clear that the ODE system is stiff (mainly due to the equations for the respiratory chain). First simulations with a stiff ODE code provided within a well-known commercial package could only be performed for up to 5 cells, which already required several minutes per run (to be multiplied by a factor of 1000 in the parameter study!). Simulations for the large systems just failed due to excess of storage requirement. The computational bottleneck turned out to be the solution of the large linear systems at each time step. In order to cope with this difficulty, the equations in the ODE system were reordered resulting in a banded structure of the Jacobian matrix: Metabolites and compartments related to individual cells were arranged within one block, as opposed to the original implementation that distributed elements over the whole system matrix. As a consequence, a numerical band solver could be applied to solve the linear equations. Upon combining the two FORTRAN codes LIMEX for stiff integration (see Sect. 2.4.2) and MUMPS, a direct parallel sparse solver (due to [1, 2], here in band mode), the ODE system could be integrated within around 10 sec. This brought the whole parameter study within a tolerable region of computing time.

After the work reported in [42], the authors further developed their model so that today the ODE system is about three times larger than the one in the original publication and heavily relies on parameter studies. In order to further speed up computations, the present even larger system is integrated with the package RoadRunner, a .NET library for carrying out numerical simulations directly from given SBML models. RoadRunner uses the (implicit) BDF integrator CVODE [37, 38] (written in C) for differential equation solving and event handling. With this change, the simulation times of the larger ODE system could be reduced to about 3 sec. per run. Moreover, a modified scientific analysis now requires about 10.000 runs per parameter study.

If, however, sensitivity studies should be included, then the linearly implicit extrapolation codes like LIMEX or METAN1 (see Sect. 2.4.2) would again enter the game.