# 8 Numerical Solution of Ordinary Differential Equations

## 8.1 Existence and Uniqueness

Please review the material in Section 8.1 of the textbook on existence and uniqueness of solutions of initial value problems for ordinary differential equations (ODEs).

## 8.2 Taylor Series Methods

We will consider general first-order initial value problems (IVPs) of the following form:

$$\begin{aligned} x'(t) &= f(t, x(t)), \qquad t \geq t_0 \\ x(t_0) &= x_0. \end{aligned} \tag{1}$$

It is our goal to derive numerical methods for the solution of this kind of problem. The first, and probably best known, method is known as *Euler's method*.

The method is derived by considering the approximation

$$x'(t) \approx \frac{x(t+h) - x(t)}{h}$$

of the first derivative. This implies

$$x(t+h) \approx x(t) + hx'(t),$$

which – using the differential equation (1) – becomes

$$x(t+h) \approx x(t) + hf(t, x(t)). \tag{2}$$

This immediately leads to an iterative algorithm.

**Algorithm**

Input $t_0$, $x_0$, $f$, $h$, $N$

$t = t_0$, $x = x_0$

for $i = 1$ to $N$ do

$x = x + hf(t, x)$

$t = t + h$

end

**Remarks:**

1. Note that Euler's method yields a set of discrete points $(t_i, x_i)$, $i = 1, \ldots, N$, which approximate the graph of the solution $x = x(t)$. In order to obtain a continuous solution one of the interpolation or approximation methods of Chapter 6 must be used.

2. Euler's method is illustrated in the Maple worksheet `578_Euler_Taylor.mws`.

An immediate generalization of Euler's method are the so-called general *Taylor series methods*. We use a Taylor expansion

$$x(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + \frac{h^3}{6}x'''(t) + \ldots,$$

and therefore obtain the numerical approximation

$$x(t+h) \approx \sum_{k=0}^{n} \frac{h^k x^{(k)}(t)}{k!} \tag{3}$$

which is referred to as an $n$-th order Taylor series method.

**Remarks:**

1. Obviously, Euler's method is a first-order Taylor method.
2. In order to program a Taylor method we need to precompute all higher-derivatives of $x$ required by the method since the differential equation only provides a representation for $x'$. This implies that we will end up with code that changes with the IVP to be solved.
3. Computer software with symbolic manipulation capabilities (such as Maple or Mathematica) allows us to write code for Taylor methods for arbitrary IVPs.

We illustrate the traditional treatment of a second-order Taylor method in the following example.

**Example:** We consider the initial value problem

$$\begin{aligned} x'(t) &= x(t) - t^2 + 1 \\ x(0) &= \frac{1}{2}. \end{aligned}$$

The second-order Taylor approximation is given by

$$x(t+h) \approx x(t) + hx'(t) + \frac{h^2}{2}x''(t).$$

Therefore, we need to express $x'(t)$ and $x''(t)$ in terms of $x$ and $t$ so that an iterative algorithm can be formulated.

From the differential equation

$$x'(t) = f(t, x(t)) = x(t) - t^2 + 1.$$

Therefore, differentiating this relation,

$$x''(t) = x'(t) - 2t,$$

and this can be incorporated into the following algorithm.

**Algorithm**

Input $t_0$, $x_0$, $f$, $h$, $N$

$t = t_0$, $x = x_0$

for $i = 1$ to $N$ do

$x' = f(t, x)$
$x'' = x' - 2t$
$x = x + hx' + \frac{h^2}{2}x''$
$t = t + h$

end

**Remarks:**

1. Two modifications are suggested to make the algorithm more efficient and numerically stable.

   (a) Replace the computation of $x$ by the nested formulation

   $$x = x + h\left(x' + \frac{h}{2}x''\right).$$

   (b) Advance the time $t$ via $t = t_0 + ih$.

2. An example of a fourth-order Taylor method is given in the Maple worksheet `578_Euler_Taylor.mws`.

**Errors**

When considering errors introduced using the Taylor series approximation we need to distinguish between two different types of error:

- local truncation error, and
- global truncation error.

The local truncation error is the error introduced directly by truncation of the Taylor series, i.e., *at each time step* we have an error

$$E_n = \frac{h^{n+1}}{(n+1)!}x^{(n+1)}(t + \theta h), \qquad 0 < \theta < 1.$$

Thus, the $n$-th order Taylor method has an $\mathcal{O}(h^{n+1})$ local truncation error.

The global truncation error is the error that results if we use an $n$-th order Taylor method having $\mathcal{O}(h^{n+1})$ local truncation error to solve our IVP up to time $t = T$. Since we will be performing

$$N = \frac{T - t_0}{h}$$

steps we see that one order of $h$ is lost in the global truncation error, i.e., the global truncation error is of the order $\mathcal{O}(h^n)$.

**Remark:** Of course, truncation errors are independent of roundoff errors which can add to the overall error.

Often we want to choose the stepsize $h$ so that the local truncation error is not too large. A rough estimate that can easily be incorporated into the algorithm is given by

$$E_n \approx \frac{h^n}{(n+1)!}\left(x^{(n)}(t+h) - x^{(n)}(t)\right)$$

(see computer homework problem 8.2#19).

## 8.3 Runge-Kutta Methods

Runge-Kutta methods are motivated by the dependence of the Taylor methods on the specific IVP. These methods do not require derivatives of the right-hand side function $f$ in the code, and are therefore general-purpose initial value problem solvers. Runge-Kutta methods are among the most popular ODE solvers. They were first studied by Carle Runge and Martin Kutta around 1900.

### 8.3.1 Second-Order Runge-Kutta Methods

We consider the general first-order ODE

$$x'(t) = f(t, x(t)). \tag{4}$$

Since we want to construct a second-order method, we start with the Taylor expansion

$$x(t+h) = x(t) + hx'(t) + \frac{h^2}{2}x''(t) + \mathcal{O}(h^3).$$

The first derivative can be replaced by the right-hand side of the differential equation (4), and the second derivative is obtained by differentiating (4), i.e.,

$$\begin{aligned} x''(t) &= f_t(t,x) + f_x(t,x)x'(t) \\ &= f_t(t,x) + f(t,x)f_x(t,x), \end{aligned}$$

where we will from now on neglect to write the dependence of $x$ on $t$ when it appears as an argument to $f$. Therefore, the Taylor expansion becomes

$$\begin{aligned} x(t+h) &= x(t) + hf(t,x) + \frac{h^2}{2}\left[f_t(t,x) + f(t,x)f_x(t,x)\right] + \mathcal{O}(h^3) \\ &= x(t) + \frac{h}{2}f(t,x) + \frac{h}{2}\left[f(t,x) + hf_t(t,x) + hf(t,x)f_x(t,x)\right] + \mathcal{O}(h^3). \end{aligned} \tag{5}$$

Recalling the multivariate Taylor expansion

$$f(x+h, y+k) = f(x,y) + hf_x(x,y) + kf_y(x,y) + \ldots$$

we see that the expression in brackets in (5) can be interpreted as

$$f(t+h, x+hf(t,x)) = f(t,x) + hf_t(t,x) + hf(t,x)f_x(t,x) + \mathcal{O}(h^2).$$

Therefore, we get

$$x(t+h) = x(t) + \frac{h}{2}f(t,x) + \frac{h}{2}f(t+h, x+hf(t,x)) + \mathcal{O}(h^3)$$

or

$$x(t+h) \approx x(t) + \frac{1}{2}\left(F_1 + F_2\right), \tag{6}$$

with

$$\begin{aligned} F_1 &= hf(t,x), \\ F_2 &= hf(t+h, x+F_1). \end{aligned}$$

This is the *classical second-order Runge-Kutta method*. It is also known as *Heun's method* or the *improved Euler method*.

We obtain general second-order Runge-Kutta methods by assuming

$$x(t+h) = x(t) + w_1F_1 + w_2F_2 + \mathcal{O}(h^3) \tag{7}$$

with

$$\begin{aligned} F_1 &= hf(t,x) \\ F_2 &= hf(t+\alpha h, x+\beta F_1). \end{aligned}$$

Clearly, this is a generalization of the classical Runge-Kutta method since the choice $w_1 = w_2 = \frac{1}{2}$ and $\alpha = \beta = 1$ yields that case.

We now need to study what other combinations of $w_1$, $w_2$, $\alpha$ and $\beta$ in (7) give us a second-order method. The bivariate Taylor expansion yields

$$\begin{aligned} f(t+\alpha h, x+\beta F_1) &= f(t,x) + \alpha h f_t(t,x) + \beta F_1 f_x(t,x) + \mathcal{O}(h^2) \\ &= f(t,x) + \alpha h f_t(t,x) + \beta h f(t,x) f_x(t,x) + \mathcal{O}(h^2). \end{aligned}$$

Therefore, the general second-order Runge-Kutta assumption (7) becomes

$$\begin{aligned} x(t+h) &= x(t) + w_1 hf(t,x) + w_2 h\left[f(t,x) + \alpha h f_t(t,x) + \beta h f(t,x) f_x(t,x)\right] + \mathcal{O}(h^3) \\ &= x(t) + (w_1+w_2)hf(t,x) + w_2h^2\left[\alpha f_t(t,x) + \beta f(t,x) f_x(t,x)\right] + \mathcal{O}(h^3). \end{aligned}$$

In order for this to match the general Taylor expansion (5) we want

$$\begin{aligned} w_1 + w_2 &= 1 \\ \alpha w_2 &= \tfrac{1}{2} \\ \beta w_2 &= \tfrac{1}{2}. \end{aligned}$$

Thus, we have a system of three nonlinear equations for our four unknowns. One popular solution is the choice $w_1 = 0$, $w_2 = 1$, and $\alpha = \beta = \frac{1}{2}$. This is known as the *modified Euler method* or the *midpoint rule*.

**Remark:** The choice $w_1 = 1$, $w_2 = 0$ leads to Euler's method. However, since now $\alpha w_2 \neq \frac{1}{2}$ and $\beta w_2 \neq \frac{1}{2}$ this method does not have second-order accuracy.

### 8.3.2 Fourth-Order Runge-Kutta Methods

The classical method (whose derivation is similar to that of the second-order method – just more technical) is given by

$$x(t+h) = x(t) + \frac{1}{6}\left[F_1 + 2F_2 + 2F_3 + F_4\right] \tag{8}$$

with

$$\begin{aligned} F_1 &= hf(t,x) \\ F_2 &= hf\left(t+\frac{h}{2}, x+\frac{1}{2}F_1\right) \\ F_3 &= hf\left(t+\frac{h}{2}, x+\frac{1}{2}F_2\right) \\ F_4 &= hf\left(t+h, x+F_3\right). \end{aligned}$$

The local truncation error for this method is $\mathcal{O}(h^5)$. It is also important to note that the classical fourth-order Runge-Kutta method requires four evaluations of the function $f$ per time step.

**Remark:** The classical fourth-order Runge-Kutta method is illustrated in the Maple worksheet `578_Runge_Kutta.mws`.

Before we consider improving the efficiency of Runge-Kutta methods we illustrate their connection to numerical integration rules.

If we consider the IVP

$$\begin{aligned} x'(t) &= f(t,x(t)) \\ x(t_0) &= x_0 \end{aligned}$$

we can separate variables

$$dx = f(t,x(t))dt$$

and then integrate both sides of this equation from $t$ to $t+h$, i.e.,

$$\int_t^{t+h} dx = \int_t^{t+h} f(\tau, x(\tau))d\tau.$$

Evaluation of the left-hand integral yields

$$x(t+h) - x(t) = \int_t^{t+h} f(\tau, x(\tau))d\tau. \tag{9}$$

Therefore, the solution to our IVP can be obtained by solving the integral equation (9). Of course, we can use numerical integration to do this:

1. Using the left endpoint method

$$\int_a^b f(x)dx \approx \underbrace{\frac{b-a}{n}}_{=h} \sum_{i=0}^{n-1} f(x_i)$$

on a single interval, i.e., with $n = 1$, and $a = t$ we get

$$\begin{aligned}\int_t^{t+h} f(\tau, x(\tau))d\tau &\approx \frac{t+h-t}{1} f(\tau_0, x(\tau_0)) \\ &= hf(t, x(t)).\end{aligned}$$

Thus, (9) is equivalent to Euler's method.

2. Using the trapezoid rule

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left[f(a) + f(b)\right]$$

with $a = t$ and $b = t + h$ gives us

$$\int_t^{t+h} f(\tau, x(\tau))d\tau \approx \frac{h}{2}\left[f(t, x(t)) + f(t+h, x(t+h))\right].$$

The corresponding IVP solver is therefore

$$x(t+h) \approx x(t) + \frac{h}{2}f(t, x(t)) + \frac{h}{2}f(t+h, x(t+h)).$$

Note that this is *not* equal to the classical second-order Runge-Kutta method since we have an $x(t+h)$ term on the right-hand side. This means that we have an *implicit* method. In order to make the method explicit we can use Euler's method to rewrite

$$x(t+h) = x(t) + hf(t, x(t)).$$

Then we end up with the method

$$x(t+h) \approx x(t) + \frac{h}{2}f(t, x(t)) + \frac{h}{2}f(t+h, x(t) + hf(t, x(t)))$$

or

$$x(t+h) \approx x(t) + \frac{1}{2}F_1 + \frac{1}{2}F_2$$

with

$$\begin{aligned}F_1 &= hf(t, x(t)) \\ F_2 &= hf(t+h, x(t) + F_1),\end{aligned}$$

i.e., the classical second-order Runge-Kutta method.

3. The midpoint integration rule leads to the modified Euler method (or midpoint rule).

4. Simpson's rule yields a special case of a fourth-order Runge-Kutta method (see HW problem 8.3#3).

5. Gauss quadrature leads to so-called Gauss-Runge-Kutta methods.

**Remark:** We saw earlier that in each time step of the second-order Runge-Kutta method we need to perform two evaluations of $f$, and for a fourth-order method there are four evaluations. More generally, one can observe the situation described in Table 1.

This data implies that higher-order ($> 4$) Runge-Kutta methods are relatively inefficient. However, certain higher-order methods may still be appropriate if we want to construct a Runge-Kutta method which adaptively chooses the step size for the time step in order to keep the local truncation error small.

| evaluations of $f$ per time step | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| maximum order achievable | 2 | 3 | 4 | 4 | 5 | 6 | 6 | 7 | 7 |

Table 1: Efficiency of Runge-Kutta methods.

### 8.3.3 Adaptive Runge-Kutta Methods

An obvious strategy for adaptive step size control is as follows: We prescribe some tolerance *tol*, and if the local truncation error $E_h$ exceeds this tolerance, then we halve the step size, i.e., $h$ is replaced by $\frac{h}{2}$. If on the other hand $E_h \ll tol$, then we double the step size.

The main question is now how to estimate the error $E_h$. A simple (but inefficient) strategy is to compute the value $x(t+h)$ twice. Once using one step of size $h$, and then again using two steps of size $\frac{h}{2}$. The difference of these two values can then be used as an error estimate.

It turns out, however, that there is a much more efficient strategy: the so-called *embedded Runge-Kutta methods*. With an embedded Runge-Kutta method we also compute the value $x(t+h)$ twice. However, it turns out that we can design methods of different orders that use the *same* function evaluations, i.e., the function evaluations used for a certain lower-order method are embedded in a second higher-order method.

We now describe the classical fourth-fifth-order Runge-Kutta-Fehlberg method which was first published in 1970. The fourth-order method is one that uses five function evaluations at each time step. Specifically,

$$x(t+h) = x(t) + \frac{25}{216}F_1 + \frac{1408}{2565}F_3 + \frac{2197}{4104}F_4 - \frac{1}{5}F_5$$

with

$$\begin{aligned}
F_1 &= hf(t,x), \\
F_2 &= hf\left(t+\frac{h}{4}, x+\frac{1}{4}F_1\right), \\
F_3 &= hf\left(t+\frac{3}{8}h, x+\frac{3}{32}F_1+\frac{9}{32}F_2\right), \\
F_4 &= hf\left(t+\frac{12}{13}h, x+\frac{1932}{2197}F_1-\frac{7200}{2197}F_2+\frac{7296}{2197}F_3\right), \\
F_5 &= hf\left(t+h, x+\frac{439}{216}F_1-8F_2+\frac{3680}{513}F_3-\frac{845}{4104}F_4\right).
\end{aligned}$$

The associated fifth-order method is given by

$$x(t+h) = x(t) + \frac{16}{135}F_1 + \frac{6656}{12825}F_3 + \frac{28561}{56430}F_4 - \frac{9}{50}F_5 + \frac{2}{55}F_6,$$

where $F_1$–$F_5$ are the same as for the fourth-order method above, and

$$F_6 = hf\left(t+\frac{h}{2}, x-\frac{8}{27}F_1+2F_2-\frac{3544}{2565}F_3+\frac{1859}{4104}F_4-\frac{11}{40}F_5\right).$$

The local truncation error is estimated by computing the deviation of the fourth-order solution from the fifth-order result.

The advantage of this embedded method is that an adaptive fifth-order method has been constructed that uses only six function evaluations for each time step.

**Remarks:**

1. An algorithm for the adaptive RKF45 method is printed in the textbook on pages 445–446. This algorithm is also implemented in most standard software packages, e.g., in Maple the method can be employed by calling the function `dsolve` with the options `numeric` and `method=rkf45`. This is illustrated in the Maple worksheet `578_Runge_Kutta.mws`. In Matlab the method can be accessed using the function `ode45`.

2. Other embedded Runge-Kutta methods also exist. For example, a fifth-sixth-order method is due to Dormand and Prince (1980). The coefficients of this method are listed in computer problem 8.3#10.

3. As mentioned earlier, Runge-Kutta methods are quite popular. This is probably due to the fact that they have been known for a long time, and are relatively easy to program. However, for so-called *stiff* problems, i.e., problems whose solution exhibits both slow and fast variations in time, Runge-Kutta methods become very inefficient. We will discuss stiff equations later.

## 8.4 Multistep Methods

Up to now, all methods we studied were single step methods, i.e., the value $x(t+h)$ was found using information only from the previous time level $t$. Now we will consider so-called *multistep methods*, i.e., more of the history of the solution will affect the value $x(t+h)$.

Consider the first-order ODE

$$x'(t) = f(t, x(t)).$$

If we integrate from $t$ to $t+h$ we have

$$\int_t^{t+h} x'(\tau)d\tau = \int_t^{t+h} f(\tau, x(\tau))d\tau$$

or

$$x(t+h) - x(t) = \int_t^{t+h} f(\tau, x(\tau))d\tau. \tag{10}$$

We now use a quadrature formula for the remaining integral.

**Example:** We interpolate $f$ at the points $\tau = t-h$ and $\tau = t$ by a linear polynomial in Lagrange form, i.e.,

$$\begin{aligned} p(\tau) &= \frac{\tau - t}{(t-h)-t} f(t-h, x(t-h)) + \frac{\tau-(t-h)}{t-(t-h)} f(t, x(t)) \\ &= \frac{t-\tau}{h} f(t-h, x(t-h)) + \frac{\tau - t + h}{h} f(t, x(t)). \end{aligned}$$

Therefore, the integral in (10) becomes

$$\begin{aligned}\int_t^{t+h} f(\tau, x(\tau))d\tau &\approx \int_t^{t+h} p(\tau)d\tau \\ &= \int_t^{t+h} \left[\frac{t-\tau}{h} f(t-h, x(t-h)) + \frac{\tau - t + h}{h} f(t, x(t))\right] d\tau \\ &= \left[f(t-h, x(t-h))\left(-\frac{1}{2}\right)\frac{(t-\tau)^2}{h} + f(t, x(t))\frac{(\tau - t + h)^2}{2h}\right]_t^{t+h} \\ &= \frac{3h}{2} f(t, x(t)) - \frac{h}{2} f(t-h, x(t-h)).\end{aligned}$$

Thus (10) gives us

$$x(t+h) \approx x(t) + \frac{h}{2}\left[3f(t, x(t)) - f(t-h, x(t-h))\right]. \tag{11}$$

If we introduce the notation $t_{n+i} = t + ih$, $i = -1, 0, 1$, and $x_{n+i} = x(t_{n+i})$, as well as $f_n = f(t_n, x_n)$, then (11) becomes

$$x_{n+1} \approx x_n + \frac{h}{2}\left[3f_n - f_{n-1}\right]. \tag{12}$$

This method is known as second-order *Adams-Bashforth method* dating back to 1883.

**Remarks:**

1. We will show later that this method is indeed of second order accuracy.
2. Note that the method (12) requires two initial conditions. Since the IVP will give us only one initial condition, we need to precede the Adams-Bashforth method by one step of, e.g., a second-order Runge-Kutta method.
3. Since (12) only involves one (new) function evaluation per time step the Adams-Bashforth method is more efficient than the corresponding Runge-Kutta method. However, it is slightly more difficult to program.

**Example:** If we use a linear Lagrange interpolant to the integrand $f$ of (10) at the points $\tau = t$ and $\tau = t + h$ then we obtain

$$x(t+h) \approx x(t) + \frac{h}{2}\left[f(t, x(t)) + f(t+h, x(t+h))\right]$$

or

$$x_{n+1} = x_n + \frac{h}{2}\left[f_n + f_{n+1}\right]. \tag{13}$$

This method is known as second-order *Adams-Moulton method*.

**Remark:** Note that the Adams-Moulton method is an *implicit* method. Such methods can be dealt with iteratively, e.g., using fixed-point iteration (see page 552 of the textbook for some more details).

Instead of dealing with the implicit Adams-Moulton method by itself one usually combines the two Adams methods into a so-called *predictor-corrector* scheme. The procedure is as follows:

1. Obtain initial values for the Adams-Bashforth method (e.g., using a Runge-Kutta method of the same order).

2. Use the Adams-Bashforth method to "predict" $x_{n+1}$.

3. Use the Adams-Moulton method to "correct" the value $x_{n+1}$ predicted by the Adams-Bashforth method.

**Remarks:**

1. Higher-order Adams methods can be derived analogously using higher-order polynomial interpolants.

2. Alternatively, the method of *undetermined coefficients* can be used. Here one assumes that the $(m+1)$-st order Adams-Bashforth method is of the form

$$x_{n+1} \approx x_n + a_0 f_n + a_1 f_{n-1} + \ldots + a_m f_{n-m}$$

and then determines the coefficients $a_0, \ldots, a_m$ by forcing the quadrature $a_0 f_n + a_1 f_{n-1} + \ldots + a_m f_{n-m}$ to be exact for polynomials of degree $m$. An example for the case $m = 4$ is listed in the textbook on page 550.

3. $(m+1)$-st order Adams-Moulton methods can be derived analogously.

**General Linear $k$-Step Methods**

The Adams-Bashforth method derived above is a two-step method, as it takes into consideration two steps back in the history of the solution. Now we will study general $k$-step methods of the form

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_0 x_{n-k} = h \left[ b_k f_n + b_{k-1} f_{n-1} + \ldots + b_0 f_{n-k} \right], \tag{14}$$

where as before $x_i = x(t_i)$, and $t_i = t_0 + ih$.

**Remarks:**

1. Note the shift in the index, i.e., the new approximate solution is no longer denoted by $x_{n+1}$, instead by $x_n$.

2. Note that both explicit (when $b_k = 0$) and implicit ($b_k \neq 0$) methods are covered by this framework.

In order to simplify the notation we will introduce a linear functional $L$ such that

$$Lx = \sum_{i=0}^{k} \left[ a_i x(ih) - h b_i \underbrace{x'(ih)}_{= f(ih, x(ih))} \right]. \tag{15}$$

For simplicity we now also assume that time is shifted so that $t_{n-k} = t_0$. With this assumption (14) is equivalent to

$$Lx = 0.$$

Note that $x$ here denotes the true solution to the discrete difference equation (14), and an approximate solution to the ODE $x'(t) = f(t, x(t))$.

If we express the values $x(ih)$ and $x'(ih)$ by their Taylor series, i.e.,

$$\begin{aligned} x(ih) &= \sum_{j=0}^{\infty} \frac{(ih)^j}{j!} x^{(j)}(0), \\ x'(ih) &= \sum_{j=0}^{\infty} \frac{(ih)^j}{j!} x^{(j+1)}(0), \end{aligned}$$

then we can rewrite (15) as

$$Lx = d_0 x(0) + d_1 h x'(0) + d_2 h^2 x''(0) + \ldots \tag{16}$$

with appropriate coefficients $d_j$. In fact, comparing (15) and (16) we see

$$\begin{aligned} d_0 &= \sum_{i=0}^{k} a_i, \\ d_1 &= \sum_{i=0}^{k} (i a_i - b_i), \\ &\vdots \\ d_j &= \sum_{i=0}^{k} \left( \frac{i^j}{j!} a_i - \frac{i^{j-1}}{(j-1)!} b_i \right), \qquad j \geq 1. \end{aligned}$$

Here we use the notational conventions $0! = 1$ and $i^0 = 1$ for any $i$.

**Characterization of Order and Accuracy**

**Theorem 8.1** *Consider the $k$-step method*

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_0 x_{n-k} = h \left[ b_k f_n + b_{k-1} f_{n-1} + \ldots + b_0 f_{n-k} \right].$$

*The following are equivalent:*

*(1) $d_0 = d_1 = \ldots = d_m = 0$,*

*(2) $Lp = 0$ for all polynomials of degree at most $m$, i.e., $L$ is exact for polynomials of degree at most $m$,*

*(3) $Lx$ is $\mathcal{O}(h^{m+1})$ for all $x \in C^{m+1}$.*

**Proof:**

(1) $\Longrightarrow$ (2)**:** Since $d_0 = d_1 = \ldots = d_m = 0$ by assumption, (16) gives us

$$Lx = d_{m+1} h^{m+1} x^{(m+1)}(0) + \ldots.$$

For (2) we consider $x$ to be a polynomial of degree at most $m$, i.e.,

$$x^{(j)}(t) = 0 \quad \text{for } j > m.$$

But then $Lx = 0$.

(2) $\Longrightarrow$ (3)**:** Now we may assume $x \in C^{m+1}$ so that it has Taylor expansion

$$x = p + r,$$

where $p$ is a polynomial of degree at most $m$, and the remainder $r$ satisfies $r^{(j)} = 0$ for $j \leq m$.

Now, using (16),

$$\begin{aligned} Lx &= \underbrace{Lp}_{=0 \text{ by (2)}} + Lr \\ &= d_{m+1}h^{m+1}r^{(m+1)}(0) + d_{m+2}h^{m+2}r^{(m+2)}(0) + \ldots \\ &= \mathcal{O}(h^{m+1}). \end{aligned}$$

(3) $\Longrightarrow$ (1)**:** If $Lx$ is of the order $\mathcal{O}(h^{m+1})$ then (16) implies

$$Lx = d_{m+1}h^{m+1}x^{(m+1)}(0) + \ldots$$

so that $d_0 = d_1 = \ldots = d_m = 0$ or (1). ♠

**Definition 8.2** *The* order *of a k-step method is the smallest integer m such that*

$$d_0 = d_1 = \ldots = d_m = 0 \neq d_{m+1}.$$

**Example:** We will show that the Adams-Bashforth method (12) derived earlier is indeed of second order. The iteration formula is (with adjusted indices)

$$x_n - x_{n-1} = \frac{h}{2}\left[3f_{n-1} - f_{n-2}\right]$$

so that $a_0 = 0$, $a_1 = -1$, $a_2 = 1$, and $b_0 = -\frac{1}{2}$, $b_1 = \frac{3}{2}$, and $b_2 = 0$ (which shows that this is an explicit method).

Now,

$$\begin{aligned} d_0 &= \sum_{i=0}^{2} a_i \\ &= 0 - 1 + 1 = 0, \end{aligned}$$

$$\begin{aligned} d_1 &= \sum_{i=0}^{2} (ia_i - b_i) \\ &= \left(0(0) + \frac{1}{2}\right) + \left((1)(-1) - \frac{3}{2}\right) + (2(1) - 0) = 0, \end{aligned}$$

$$\begin{aligned} d_2 &= \sum_{i=0}^{2} \left(\frac{i^2}{2}a_i - ib_i\right) \\ &= 0 + \left(\frac{1}{2}(-1) - \frac{3}{2}\right) + \left(\frac{4}{2}(1) - (2)(0)\right) = 0, \end{aligned}$$

$$\begin{aligned} d_3 &= \sum_{i=0}^{2} \left( \frac{i^3}{6} a_i - \frac{i^2}{2} b_i \right) \\ &= 0 + \left( \frac{1}{6}(-1) - \frac{1}{2}\frac{3}{2} \right) + \left( \frac{8}{6}(1) - \frac{4}{2}0 \right) \\ &= \frac{5}{12} \neq 0. \end{aligned}$$

Therefore, the method is indeed of order 2.

**Remark:** In principle, the conditions of Definition 8.2 can be used to derive $k$-step methods of order $2k$. However, as we will see below, a numerical method also needs to be *stable*. It is known that stable $k$-step methods can be of order at most $k + 2$.

## 8.5 Errors and Stability

We now denote by $x(h, t)$ the value of the approximate solution at time $t$ based on computations with time steps of length $h$, and let $x(t)$ denote the value of the true solution at time $t$ to our initial value problem $x'(t) = f(t, x(t))$ with initial condition $x(t_0) = x_0$.

**Definition 8.3** *A $k$-step method is called* convergent *if*

$$\lim_{h \to 0} x(h, t) = x(t)$$

*for all (fixed) $t$ in some interval $[t_0, t_m]$ provided*

$$\lim_{h \to 0} x(h, t_0 + ih) = x_0$$

*for all $0 \leq i < k$.*

For the following discussion we identify two polynomials associated with the $k$-step method

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_0 x_{n-k} = h \left[ b_k f_n + b_{k-1} f_{n-1} + \ldots + b_0 f_{n-k} \right].$$

Let $p$ and $q$ be defined via

$$\begin{aligned} p(z) &= \sum_{i=0}^{k} a_i z^i \\ q(z) &= \sum_{i=0}^{k} b_i z^i \end{aligned}$$

with the coefficients $a_i$ and $b_i$ given above.

**Definition 8.4** *A $k$-step method is called* zero stable *if all roots $\lambda$ of the polynomial $p$ lie inside the unit disk, i.e., $|\lambda| \leq 1$, and those on the unit circle are simple, i.e., if $|\lambda| = 1$ then $p'(\lambda) \neq 0$.*

**Definition 8.5** *A $k$-step method is called* consistent *if it is exact for constants and linear polynomials, i.e., $d_0 = d_1 = 0$.*

**Remarks:**

1. Consistency of a $k$-step method can also be characterized by $p(1) = 0$ and $p'(1) = q(1)$.
2. Theorem 8.1 implies that any first-order method is consistent.

The central theorem known as the *Lax Equivalence Theorem* is

**Theorem 8.6** *A $k$-step method is convergent if and only if it is zero stable and consistent.*

**Proof:** The fact that zero stability and consistency are sufficient for convergence is rather involved. We do not discuss it here. Necessity of zero stability and consistency are somewhat easier. We concentrate on necessity of zero stability (consistency is covered in the textbook on page 559).

By assumption the method is convergent for all admissible right-hand side functions $f$, and therefore it is also convergent for the particular IVP with $f \equiv 0$ and zero initial data, i.e.,

$$\begin{aligned} x'(t) &= 0 \\ x(0) &= 0. \end{aligned}$$

Therefore, by assumption, the numerical scheme must converge to the zero solution for any set of initial conditions that satisfy the second part of Definition 8.3.

We will use a proof by contradiction, and show that if the method is not zero stable, then the above convergence claim does not hold.

If the method is not zero stable, then either some root $\lambda$ of $p$ lies outside the unit disk, or is on the unit circle but has multiplicity greater than one, i.e.,

$$|\lambda| > 1 \quad \text{or} \quad |\lambda| = 1 \text{ with } p'(\lambda) = 0.$$

For the simple problem above the $k$-step method becomes

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_0 x_{n-k} = 0. \tag{17}$$

In the case $|\lambda| > 1$ we see that $x_n = h\lambda^n$ is a possible solution since (17) is equivalent to

$$\begin{aligned} & a_k h\lambda^n + a_{k-1} h\lambda^{n-1} + \ldots a_0 h\lambda^{n-k} \\ &= h\lambda^{n-k} \left[ a_k \lambda^k + a_{k-1} \lambda^{k-1} + \ldots + a_0 \right] \\ &= h\lambda^{n-k} \underbrace{p(\lambda)}_{=0} = 0. \end{aligned}$$

Now, for any fixed $\lambda$ and $0 \le n < k$,

$$|x(h, nh)| = |x_n| = h|\lambda^n| < h|\lambda|^k \to 0$$

for $h \to 0$. This establishes the second part of Definition 8.3. However, we can see that the first part cannot be satisfied by considering, for any $t = nh$,

$$|x(h, nh)| = |x_n| = h|\lambda^n| = \frac{t}{n}|\lambda|^n.$$

This quantity tends to $\infty$ for $n \to \infty$ ($\Longleftrightarrow h \to 0$ since $h = \frac{t}{n}$, $t$ fixed). So there is no convergence in this case.

If $|\lambda| = 1$ with $p'(\lambda) = 0$ then $x_n = hn\lambda^n$ is another possible solution of (17) since

$$\begin{aligned}
&a_k hn\lambda^n + a_{k-1}h(n-1)\lambda^{n-1} + \ldots a_0 h(n-k)\lambda^{n-k} \\
&= h\lambda^{n-k}\left[\underbrace{n}_{=n-k+k} a_k\lambda^k + \underbrace{(n-1)}_{=n-k+k-1} a_{k-1}\lambda^{k-1} + \ldots (n-k)a_0\right] \\
&= h\lambda^{n-k}\left\{(n-k)\left[a_k\lambda^k + a_{k-1}\lambda^{k-1} + \ldots + a_0\right] + \left[ka_k\lambda^k + (k-1)a_{k-1}\lambda^{k-1} + \ldots + 0\cdot a_0\right]\right\} \\
&= h\lambda^{n-k}\left[(n-k)\underbrace{p(\lambda)}_{=0} + \lambda\underbrace{p'(\lambda)}_{=0}\right] = 0.
\end{aligned}$$

Again, the initial values satisfy the second part of Definition 8.3, i.e., for $0 \le n < k$ (with $k$ fixed)

$$|x(h, nh)| = |x_n| = hn\underbrace{|\lambda^n|}_{=1} = hn < hk \to 0$$

for $h \to 0$. On the other hand, the first part fails again since

$$|x(h, nh)| = |x_n| = hn|\lambda^n| = hn = \frac{t}{n}n = t \neq 0.$$

♠

**Example:** We already know that the second-order Adams-Bashforth method is consistent. Now we show it is also zero stable, and therefore convergent.

Since

$$x_n - x_{n-1} = \frac{h}{2}\left[3f_{n-1} - f_{n-2}\right]$$

we have $a_0 = 0$, $a_1 = -1$, $a_2 = 1$, and

$$p(z) = z^2 - z = z(z-1).$$

The roots of $p$ are $\lambda = 0$ and $\lambda = 1$. Thus, all roots lie within the unit disk, and the root on the boundary is simple.

**Local Truncation Error**

One can show (see textbook page 561)

**Theorem 8.7** *If a $k$-step method is of order $m$, $x \in C^{m+2}$ and $\frac{\partial f}{\partial x} \in C$, then the local truncation error*

$$x(t_n) - x_n = \mathcal{O}(h^{m+1})$$

*provided the earlier values $x_{n-1}$, $x_{n-2}$, ..., $x_{n-k}$ were computed exactly.*

**Global Truncation Error**

For multistep methods the global truncation error is not simply the sum of local truncation errors. The following concept of stability also plays an important role in determining the global truncation error.

**Definition 8.8** *A numerical IVP solver is* stable *if small perturbations in the initial conditions do not cause the numerical approximation to diverge away from the true solution provided the true solution of the initial value problem is bounded.*

**Remark:** For consistent $k$-step methods one can show that the notions of stability and zero stability are equivalent.

**Example:** For $\lambda \in \mathbb{R}$ we consider the family of initial value problems

$$\begin{aligned} x'(t) &= \lambda x(t), \quad 0 \le t \le T \\ x(0) &= x_0. \end{aligned}$$

This is a linear equation. The solution of this problem is given by

$$x(t) = x_0 e^{\lambda t}.$$

Now we take the same differential equation, but with perturbed initial condition

$$x_\delta(0) = x_0 + \delta.$$

Then the general solution still is $x_\delta(t) = ce^{\lambda t}$. However, the initial condition now implies

$$x_\delta(t) = (x_0 + \delta)e^{\lambda t}.$$

Therefore, if $\lambda \le 0$, a small change in the initial condition causes only a small change in the solution and therefore the problem is a stable problem. However, if $\lambda > 0$, then large changes in the solution will occur (even for small perturbations of the initial condition), and the problem is unstable.

A stable numerical method is one for which the numerical solution of a stable problem behaves also in this controlled fashion.

**Example:** We study how Euler's method (a one-step method) behaves for the stable problem of the previous example, i.e., in the case $\lambda \le 0$. Euler's method states that

$$\begin{aligned} x_n &= x_{n-1} + hf_{n-1} \\ &= x_{n-1} + h\lambda x_{n-1} \\ &= (1 + \lambda h)x_{n-1}. \end{aligned}$$

Therefore,

$$x_n = (1 + \lambda h)^n x_0.$$

Since the exact problem has an exponentially decaying solution for $\lambda < 0$, a stable numerical method should exhibit the same behavior. Therefore, in order to ensure

stability of Euler's method we need that the so-called *growth factor* $|1 + \lambda h| < 1$. For $\lambda < 0$ this is equivalent to

$$-2 < h\lambda < 0 \quad \Longleftrightarrow \quad h < \frac{-2}{\lambda}.$$

Thus, Euler's method is only *conditionally stable*, i.e., the step size has to be chosen sufficiently small to ensure stability.

**Example:** On the other hand, we can show that the implicit or *backward Euler* method

$$x_n = x_{n-1} + hf_n$$

is *unconditionally stable* for the above problem.

To see this we have

$$x_n = x_{n-1} + h\lambda x_n$$

or

$$(1 - \lambda h)x_n = x_{n-1}.$$

Therefore,

$$x_n = \left(\frac{1}{1 - \lambda h}\right)^n x_0.$$

Now, for $\lambda < 0$, the growth factor

$$\left(\frac{1}{1 - \lambda h}\right) < 1$$

for any $h > 0$, and we can choose the step size $h$ arbitrarily large.

**Remarks:**

1. The backward Euler can be shown to be unconditionally stable for arbitrary first-order IVPs.

2. The backward Euler method is only a first-order method, and therefore it is of limited accuracy and of not much practical use for general problems. Moreover, it is an implicit method, and therefore more complicated to program.

3. However, the backward Euler method is a rather popular solver for stiff problems.

More generally, we consider the initial value problem

$$\begin{aligned} x'(t) &= f(t, x(t)) \\ x(0) &= s \end{aligned}$$

and we want to understand how changes in the initial condition $s$ affect the solution $x(t; s)$ at time $t$.

To this end we define

$$u(t) = \frac{\partial}{\partial s} x(t; s)$$

and consider the *variational equation*

$$\frac{\partial}{\partial s}\left[x'(t) = f(t, x(t; s))\right] \quad \Longleftrightarrow \quad u'(t) = f_x(t, x)x_s(t; s).$$

If we add the initial condition

$$\frac{\partial}{\partial s}\left[x(0;s) = s\right] \quad \Longleftrightarrow \quad u(0) = 1,$$

then we end up with an IVP for $u$ of the form

$$\begin{aligned} u'(t) &= f_x(t,x)u(t) \\ u(0) &= 1. \end{aligned} \tag{18}$$

This is similar to the IVP studied in the Example above. In particular, if $f_x(t,x) = \lambda$ then the solution is $u(t) = e^{\lambda t}$. Therefore,

**Theorem 8.9** *If $f_x \leq \lambda$ then the solution of (18) satisfies*

$$|u(t)| \leq e^{\lambda t}.$$

*Moreover,*

$$|x(t;s) - x(t;s+\delta)| \leq |\delta| e^{\lambda t}.$$

**Theorem 8.10** *If the local truncation errors of a $k$-step method at $t_1$, $t_2$, ..., $t_n$ do not exceed $\delta$ in magnitude, then the global truncation error at $t_n$ satisfies*

$$|x(t_n) - x_n| \leq \delta \frac{e^{n\lambda h} - 1}{e^{\lambda h} - 1}.$$

**Proof:** Denote the individual local truncation errors by

$$\delta_j = |x(t_j) - x_j|, \quad j = 1, \ldots, n.$$

We now construct the global truncation error in a iterative manner. The main ingredient is Theorem 8.9. Then

$$\begin{aligned} |x(t_1) - x_1| &= \delta_1 \\ |x(t_2) - x_2| &\leq |\delta_1| e^{\lambda h} + |\delta_2| \\ |x(t_3) - x_3| &\leq \left(|\delta_1| e^{\lambda h} + |\delta_2|\right) e^{\lambda h} + |\delta_3| \\ &\vdots \\ |x(t_n) - x_n| &\leq \sum_{j=1}^{n} |\delta_j| e^{(n-j)\lambda h}. \end{aligned}$$

Since, for all $j$, $|\delta_j| \leq \delta$ we can simplify the estimate for the global truncation error

$$\begin{aligned} |x(t_n) - x_n| &\leq \delta \sum_{j=1}^{n} e^{(n-j)\lambda h} \\ &= \delta \left[e^{(n-1)\lambda h} + e^{(n-2)\lambda h} + \ldots + e^{0}\right] \\ &= \delta \sum_{j=0}^{n-1} e^{j\lambda h}. \end{aligned}$$

However, this is a finite geometric series, and the desired estimate follows. ♠

**Corollary 8.11** *If all local truncation errors of a k-step method are $\mathcal{O}(h^{m+1})$ then the global truncation error is $\mathcal{O}(h^m)$.*

**Proof:** We apply Theorem 8.10. Since $t = nh$ we have

$$\frac{e^{n\lambda h} - 1}{e^{\lambda h} - 1} = \frac{e^{\lambda t} - 1}{e^{\lambda h} - 1}.$$

Moreover, the Taylor expansion of the exponential function gives us

$$e^{\lambda h} - 1 = \mathcal{O}(\lambda h).$$

Therefore, with $\delta = \mathcal{O}(h^{m+1})$, Theorem 8.10 says

$$|x(t_n) - x_n| \leq \mathcal{O}(h^{m+1}) \frac{e^{\lambda t} - 1}{\mathcal{O}(\lambda h)} = \mathcal{O}(h^m).$$

♠

## 8.6 Systems and Higher-Order ODEs

All of the methods studied so far can be adapted to systems of first-order ODEs. Consider the general first-order system

$$\begin{array}{rcl} x_1'(t) & = & f_1(t, x_1(t), x_2(t), \ldots, x_n(t)) \\ x_2'(t) & = & f_2(t, x_1(t), x_2(t), \ldots, x_n(t)) \\ & \vdots & \\ x_n'(t) & = & f_n(t, x_1(t), x_2(t), \ldots, x_n(t)) \end{array} \tag{19}$$

with $n$ unknown functions $x_1, \ldots, x_n$, and initial conditions

$$x_1(t_0) = c_1, \quad x_2(t_0) = c_2, \quad \ldots, x_n(t_0) = c_n.$$

Using the vector notation

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad F = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \quad \text{and} \quad C = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix},$$

the system (19) can be compactly written as

$$X'(t) = F(t, X(t)) \tag{20}$$

subject to the vector initial condition $X(t_0) = C$.

**Remark:** If $F(t, X(t) = F(X(t))$, i.e., there is no explicit dependence on time, then the system is called *autonomous*.

All numerical IVP solvers can be applied immediately to such autonomous systems as long as all operations are interpreted in a vectorized fashion. For example, the solvers included in Maple and Matlab can all handle vectorized input.

**Example:** The fourth-order Runge-Kutta method for an autonomous first-order system of the form

$$X'(t) = F(X(t))$$

is simply

$$X(t+h) = X(t) + \frac{h}{6}\left(F_1 + 2F_2 + 2F_3 + F_4\right),$$

where

$$\begin{aligned} F_1 &= F(X(t)) \\ F_2 &= F\left(X(t) + \frac{h}{2}F_1\right) \\ F_3 &= F\left(X(t) + \frac{h}{2}F_2\right) \\ F_4 &= F\left(X(t) + hF_3\right). \end{aligned}$$

**Example:** The *predator-prey* model gives rise to an autonomous system of nonlinear first-order equations

$$\begin{aligned} x'(t) &= -dx(t) + ax(t)y(t) \\ y'(t) &= by(t) - cx(t)y(t) \end{aligned}$$

known as a *Lotka-Volterra* system. Here the positive constants $d$ and $b$ can be interpreted as the death and birth rates of the predator ($x$) and prey ($y$), respectively. The other two positive constants, $a$ and $c$, describe the interaction between the two species. We can add initial conditions $x(0) = \alpha$, $y(0) = \beta$. For general parameters $a, b, c, d$, there is no closed-form solution to this problem. The only way to solve this problem is using a numerical solver.

In vector notation the predator-prey system is written as

$$X'(t) = F(X(t)),$$

with

$$X = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad F(X) = \begin{bmatrix} -dx + axy \\ by - cxy \end{bmatrix}.$$

We solve two Lotka-Volterra systems numerically with a fourth-order Runge-Kutta method in the Maple worksheet `578_ODESystems.mws`.

**Non-Autonomous Systems**

Non-autonomous systems can be represented as autonomous systems in a straightforward way. We illustrate the general principle with an example.

**Example:** Consider the following non-autonomous system of two first-order ODEs:

$$\begin{aligned} x_1'(t) &= x_1(t) - x_2(t) + 2t \\ x_2'(t) &= x_1(t) + x_2(t) + t^3. \end{aligned} \tag{21}$$

In order to obtain an autonomous system we introduce the auxiliary function $x_0(t) = t$ leading to the extra ODE $x_0'(t) = 1$. Then the system (21) becomes

$$\begin{aligned} x_0'(t) &= 1 \\ x_1'(t) &= x_1(t) - x_2(t) + 2x_0(t) \\ x_2'(t) &= x_1(t) + x_2(t) + [x_0(t)]^3, \end{aligned} \tag{22}$$

or in vector notation

$$X'(t) = F(X(t))$$

with

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}, \qquad F(X) = \begin{bmatrix} 1 \\ x_1 - x_2 + 2x_0 \\ x_1 + x_2 + x_0^3 \end{bmatrix}.$$

**Higher-Order Initial Value Problems**

Once we have code to solve vectorized first-order initial value problems, it is again straightforward to apply this code to the solution of higher-order initial value problems. All we need to do is rewrite the $n$-th order initial value problem as a system of $n$ first-order equations.

**Example:** Consider the second-order IVP

$$y''(t) - 2y'(t) + 2y(t) = e^{2t} \sin t, \qquad 0 \le t \le 1,$$

with initial conditions

$$y(0) = \alpha, \quad y'(0) = \beta.$$

If we let $x_1 = y$ and $x_2 = y'$, then we obtain the two first-order equations

$$\begin{aligned} x_1'(t) &= x_2(t) \\ x_2'(t) &= 2x_2(t) - 2x_1(t) + e^{2t} \sin t. \end{aligned}$$

These equations are non-autonomous. Therefore we can – as before – introduce $x_0(t) = t$, and end up with the three autonomous equations

$$\begin{aligned} x_0'(t) &= 1 \\ x_1'(t) &= x_2(t) \\ x_2'(t) &= 2x_2(t) - 2x_1(t) + e^{2x_0(t)} \sin x_0(t). \end{aligned}$$

The corresponding initial conditions are

$$x_0(0) = 0, \quad x_1(0) = \alpha, \quad x_2(0) = \beta.$$

**Systems of Higher-Order Initial Value Problems**

By combining the two ideas used above, we can also use our vectorized IVP solvers to solve systems of $n$-th order initial value problems.

**Example:** Consider for $t \ge 1$ the coupled non-autonomous system of two second-order ODEs

$$y''(t) - 2y'(t) + z(t) = te^t - t$$

$$t^2 z''(t) - 2ty'(t) + 2z(t) \quad = \quad t^3 \ln t$$

with initial conditions

$$y(1) = \alpha, \quad y'(1) = \beta, \quad z(1) = \gamma, \quad z'(1) = \delta.$$

Introducing the new variables $x_1 = y$, $x_2 = y'$, $x_3 = z$, and $x_4 = z'$ we get

$$\begin{aligned} x_1'(t) &= x_2(t) \\ x_2'(t) &= te^t - t + 2x_2(t) - x_3(t) \\ x_3'(t) &= x_4(t) \\ x_4'(t) &= t \ln t + \frac{2}{t} x_2(t) - \frac{2}{t^2} x_3(t) \end{aligned}$$

with initial conditions

$$x_1(1) = \alpha, \quad x_2(1) = \beta, \quad x_3(1) = \gamma, \quad x_4(1) = \delta.$$

Of course, the system can again be made autonomous by introducing $x_0(t) = t$.

## 8.7 Stiff ODEs

Some of this material is covered in Section 8.12 of the textbook.

Stiffness in differential equations can be interpreted in various ways. As mentioned earlier, physically, a problem

$$X'(t) = F(t, X(t))$$

is considered stiff if its solution $X$ exhibits features present on largely varying time scales. A more mathematical interpretation is that the eigenvalues of the Jacobian matrix of $F$ differ greatly in magnitude. Another way to identify a stiff ODE is if stability considerations force the step size $h$ to be much smaller than it would have to be to satisfy the desired accuracy.

**Remark:** Stiff ODEs arise in various applications; e.g., when modeling chemical reactions, or electrical circuits, such as the van der Pol equation in relaxation oscillation (see the Maple worksheet `578_StiffODEs.mws`).

We begin with an example.

**Example:** Consider for $\lambda > 0$ the second-order initial value problem

$$x''(t) + (\lambda + 1)x'(t) + \lambda x(t) = 0, \quad 0 < t < 1,$$

with initial conditions $x(0) = 1$ and $x'(0) = 0$. The analytical solution is given by

$$x(t) = \frac{\lambda}{\lambda - 1} e^{-t} - \frac{1}{\lambda - 1} e^{-\lambda t}.$$

It is easy to verify that the problem itself is a stable problem (as long as $\lambda > 0$). However, we see that for very large values of $\lambda$ the solution contains a relatively slowly

decaying part, as well as the rapidly decaying *transient* component $e^{-\lambda t}$. Therefore, for large $\lambda$ this problem is stiff.

In order to apply one of our standard initial value problem solvers we rewrite the second-order equation as a system of two first-order equations

$$\begin{aligned} x_1'(t) &= x_2(t) \\ x_2'(t) &= -\lambda x_1(t) - (\lambda+1)x_2(t) \end{aligned}$$

with initial conditions $x_1(0) = 1$ and $x_2(0) = 0$.

We illustrate the use of the standard fourth-fifth order Runge-Kutta-Fehlberg method for the problem in the Maple worksheet `578_StiffODEs.mws`.

**Remark:** We can observe that, for increased values of the parameter $\lambda$, the RKF45 method requires more and more function evaluations, and thus becomes more and more inefficient. Notice that the problem itself, on the other hand, is more and more dominated by the $e^{-t}$ term, and thus actually easier to solve.

The best way to deal with a stiff problem is to use a numerical method that is unconditionally stable. In fact, unconditional stability for the test problem

$$\begin{aligned} x'(t) &= \lambda x(t), \quad 0 \le t \le T \\ x(0) &= x_0. \end{aligned}$$

is all that is needed for an effective stiff solver. This property is usually called *A-stability*.

The *region of absolute stability* of a $k$-step method is the region of the complex plane for which the roots of the polynomial $\phi = p - \lambda h q$ lie inside the unit disk. Here $p$ and $q$ are defined as earlier, i.e.,

$$\begin{aligned} p(z) &= \sum_{i=0}^{k} a_i z^i \\ q(z) &= \sum_{i=0}^{k} b_i z^i \end{aligned}$$

and $a_i$ and $b_i$ are the coefficients of the $k$-step method

$$a_k x_n + a_{k-1} x_{n-1} + \ldots + a_0 x_{n-k} = h\left[b_k f_n + b_{k-1} f_{n-1} + \ldots + b_0 f_{n-k}\right]$$

with $f_j = \lambda x_j$, $j = n-k, \ldots, n$, given by the test problem.

**Definition 8.12** *A $k$-step method is* A-stable *if its region of absolute stability includes the entire negative real axis.*

We already have seen one $A$-stable method earlier: the backward (or implicit) Euler method

$$x_n = x_{n-1} + h f(t_n, x_n).$$

In general, only implicit multistep methods are candidates for stiff solvers. However, the following theorem (due to Dahlquist) reveals the limited accuracy that can be achieved by such $A$-stable $k$-step methods.

**Theorem 8.13** *If a linear $k$-step method is $A$-stable then it must be an implicit method. Moreover, the order of the method is at most 2.*

Theorem 8.13 implies that the only other $k$-step method we need to consider is the *implicit trapezoid method* (or second-order Adams-Moulton method) introduced at the beginning of Section 8.4:

$$x_n = x_{n-1} + \frac{h}{2}\left(f_n + f_{n-1}\right). \tag{23}$$

We have not verified that this method is of second order (but this can easily be done using Theorem 8.1). To see that the implicit trapezoid method is $A$-stable we consider

$$x_n = x_{n-1} + \frac{h}{2}\lambda\left(x_n + x_{n-1}\right)$$

or

$$(1 - \lambda\frac{h}{2})x_n = (1 + \lambda\frac{h}{2})x_{n-1}.$$

Therefore,

$$x_n = \left(\frac{1 + \lambda\frac{h}{2}}{1 - \lambda\frac{h}{2}}\right)^n x_0,$$

and, for $\lambda < 0$, the growth factor

$$\frac{1 + \lambda\frac{h}{2}}{1 - \lambda\frac{h}{2}} = \frac{2 + \lambda h}{2 - \lambda h} < 1$$

for any $h > 0$.

**Remark:** As mentioned earlier, in order to implement the implicit trapezoid method (23) it needs to be coupled with Newton iteration.

Of course, one can also try to develop methods that do not completely satisfy the condition of $A$-stability, and hope to achieve higher order by doing this. The state-of-the-art stiff solvers today seem to be one of the following two methods:

1. The so-called *Gear* methods achieve this by monitoring the largest and smallest eigenvalues of the Jacobian matrix, and thus assessing and dealing with the stiffness adaptively. Gear methods employ variable order as well as variable step size. In Maple some of these methods are available through the `method=lsode` option (invoking the Livermore ODEPACK by Hindmarsh) to `dsolve,numeric`. In Matlab, Gear methods are implemented in the stiff solver `ode15s`.

2. Another class of stiff solvers are implicit Runge-Kutta methods known as *Rosenbrock* methods. In Maple a third-fourth order Rosenbrock method is the default implementation in `dsolve,numeric` with the `stiff=true` option. In Matlab a second-third order Rosenbrock method is implemented in the routine `ode23s`.

We illustrate the use of stiff solvers in the Maple worksheet `578_StiffODEs.mws`.

References for this subject are the books "Numerical Initial Value Problems in Ordinary Differential Equations" by Bill Gear (1971), or "Solving Ordinary Differential

Equations II, Stiff and Differential-Algebraic Problems" by Hairer and Wanner (1991), or the two books "Numerical Solution of Ordinary Differential Equations" by Larry Shampine (1994) and "Computer Solution of Ordinary Differential Equations: the Initial Value Problem" by Shampine and Gordon (1975).

## 8.8 Boundary Value Problems: Theory

We now consider second-order boundary value problems of the general form

$$\begin{gathered} x''(t) = f(t, x(t), x'(t)) \\ a_0 x(a) + a_1 x'(a) = \alpha, \quad b_0 x(b) + b_1 x'(b) = \beta. \end{gathered} \tag{24}$$

**Remarks:**

1. Note that this kind of problem can no longer be converted to a system of two first order initial value problems as we did in Section 8.6.

2. Boundary value problems of this kind arise in many applications, e.g., in mechanics (bending of an elastic beam), fluids (flow through pipes, laminar flow in a channel, flow through porous media), or electrostatics.

The mathematical theory for boundary value problems is more complicated (and less well known) than for initial value problems. Therefore, we present a version of an existence and uniqueness theorem for the general problem (24).

**Theorem 8.14** *Suppose $f$ in (24) is continuous on the domain $D = \{(t, y, z) : a \leq t \leq b, -\infty < y < \infty, -\infty < z < \infty\}$ and that the partial derivatives $f_y$ and $f_z$ are also continuous on $D$. If*

1. *$f_y(t, y, z) > 0$ for all $(t, y, z) \in D$,*

2. *there exists a constant $M$ such that*

$$|f_z(t, y, z)| \leq M$$

   *for all $(t, y, z) \in D$, and*

3. *$a_0 a_1 \leq 0$, $b_0 b_1 \leq 0$, and $|a_0| + |b_0| > 0$,*

*then the boundary value problem (24) has a unique solution.*

**Proof:** A proof of this theorem can be found, e.g., in the book "Numerical Methods for Two-Point Boundary Value Problems" by H. B. Keller (1968). ♠

**Example:** Consider the BVP

$$\begin{gathered} x''(t) + e^{-tx(t)} + \sin x'(t) = 0, \quad 1 \leq t \leq 2, \\ x(1) = x(2) = 0. \end{gathered}$$

To apply Theorem 8.14 we identify $f(t, y, z) = -e^{-ty} - \sin z$. Then

$$f_y(t, y, z) = te^{-ty}$$

which is positive for all $t > 0$, $y, z \in \mathbb{R}$. So, in particular it is positive for $1 \leq t \leq 2$. Moreover, we identify $f_z(t, y, z) = -\cos z$, so that

$$|f_z(t, y, z)| = |-\cos z| \leq 1 = M.$$

Obviously, all continuity requirements are satisfied. Finally, we have $a_0 = b_0 = 1$ and $a_1 = b_1 = 0$, so that the third condition is also satisfied. Therefore, the given problem has a unique solution.

If the boundary value problem (24) takes the special form

$$\begin{aligned} x''(t) &= u(t) + v(t)x(t) + w(t)x'(t) \\ a_0x(a) + a_1x'(a) = \alpha, &\quad b_0x(b) + b_1x'(b) = \beta, \end{aligned} \tag{25}$$

then it is called *linear*. In this case Theorem 8.14 simplifies considerably.

**Theorem 8.15** *If $u, v, w$ in (25) are continuous and $v(t) > 0$ on $[a, b]$, then the linear boundary value problem (25) has a unique solution.*

**Remark:** A few more results are compiled in Section 8.7 of the textbook. A classical reference for the numerical solution of two-point BVPs is the book by Keller mentioned above. A modern reference is "Numerical Solution of Boundary Value Problems for Ordinary Differential Equations" by Ascher, Mattheij, and Russell (1995).

## 8.9 Boundary Value Problems: Shooting Methods

One of the most popular, and simplest strategies to apply for the solution of two-point boundary value problems is to convert them to sequences of initial value problems, and then use the techniques developed for those methods.

We now restrict our discussion to BVPs of the form

$$\begin{aligned} x''(t) &= f(t, x(t), x'(t)) \\ x(a) = \alpha, &\quad x(b) = \beta. \end{aligned} \tag{26}$$

With some modifications the methods discussed below can also be applied to the more general problem (24).

The fundamental idea on which the so-called *shooting methods* are based is to formulate an initial value problem associated with (26). Namely,

$$\begin{aligned} x''(t) &= f(t, x(t), x'(t)) \\ x(a) = \alpha, &\quad x'(a) = z. \end{aligned} \tag{27}$$

After rewriting this second-order initial value problem as two first-order problems we can solve this problem with our earlier methods (e.g., Runge-Kutta or $k$-step methods),

and thus obtain a solution $x_z$. In order to see how well this solution matches the solution $x$ of the two-point boundary value problem (26) we compute the difference

$$\phi(z) := x_z(b) - \beta$$

at the right end of the domain. If the initial slope $z$ was chosen correctly, then $\phi(z) = 0$ and we have solved the problem. If $\phi(z) \neq 0$, we can use a solver for nonlinear equations (such as secant or Newton iteration discussed last semester in Chapter 3) to find a better slope.

**Remarks:**

1. Changing the "aim" of the initial value problem by adjusting the initial slope to "hit" the target value $x(b) = \beta$ is what gave the name to this numerical method.

2. Even though the shooting method is fairly simple to implement, making use of standard code for initial value problems, and a nonlinear equation solver, it inherits the stability issues encountered earlier for IVP solvers. For boundary value problems the situation is even worse, since even for a stable boundary value problem, the associated initial value problem can be unstable, and thus hopeless to solve.

We illustrate the last remark with

**Example:** For $\lambda < 0$ the (decoupled) boundary value problem

$$\begin{aligned} x_1'(t) &= \lambda x_1(t) \\ x_2'(t) &= -\lambda x_2(t) \\ x_1(0) = 1, &\quad x_2(a) = 1 \end{aligned}$$

for $t \in [0, a]$ is stable since the solution $x_1(t) = e^{\lambda t}$, $x_2(t) = e^{a\lambda}e^{-\lambda t}$ remains bounded for $t \to \infty$ even for large values of $a$. On the other hand, the initial value problem

$$\begin{aligned} x_1'(t) &= \lambda x_1(t) \\ x_2'(t) &= -\lambda x_2(t) \\ x_1(0) = \alpha, &\quad x_2(0) = \beta \end{aligned}$$

is unstable for any $\lambda \neq 0$ since always one of the components of the solution $x_1(t) = \alpha e^{\lambda t}$, $x_2(t) = \beta e^{-\lambda t}$ will grow exponentially.

**Remark:** A convergence analysis for the shooting method is very difficult since two types of errors are now involved. On the one hand there is the error due to the IVP solver, and on the other hand there is the error due to the discrepancy of the solution at the right boundary.

**Using the Secant Method**

Recall that the secant method for solving the equation $\phi(z) = 0$ is

$$z_{n+1} = z_n - \phi(z_n)\frac{z_n - z_{n-1}}{\phi(z_n) - \phi(z_{n-1})}, \quad n \geq 1.$$

This method requires two starting values $z_0$ and $z_1$, i.e., two initial guesses need to be provided. In order to find the updated slope $z_{n+1}$ (i.e., when using the secant method) we need the values $\phi(z_{n-1})$ and $\phi(z_n)$. These values are rather expensive to obtain since

$$\phi(z_n) = x_{z_n}(b) - \beta,$$

and $x_{z_n}(b)$ is obtained by solving an initial value problem up to $t = b$. Thus, each improvement of the slope $z$ requires the solution of two initial value problems.

**Algorithm**

1. Provide two initial slopes $z_0$ and $z_1$, and a tolerance $\epsilon$.

2. Solve the two initial value problems (27) with initial conditions $x'(a) = z_0$ and $x'(a) = z_1$. This yields solutions $x_{z_0}$ and $x_{z_1}$. Let $n = 1$. Then

$$\phi(z_{n-1}) = x_{z_{n-1}}(b) - \beta \quad \text{and} \quad \phi(z_n) = x_{z_n}(b) - \beta.$$

3. Apply the secant method, i.e., compute

$$z_{n+1} = z_n - \phi(z_n)\frac{z_n - z_{n-1}}{x_{z_n}(b) - x_{z_{n-1}}(b)}.$$

4. Check if $|\phi(z_{n+1})| < \epsilon$. If yes, stop. Otherwise, increment $n$ and repeat from 3.

**Remarks:**

1. Note that the comparison in Step 4 involves the solution of another initial value problem.

2. Once a number of slopes $z_j$ and values $\phi(z_j)$, $j = 1, \ldots, n$, have been computed one can also use polynomial interpolation to obtain an updated value for the slope. To this end one constructs a polynomial $p$ that interpolates $p(\phi(z_j)) = z_j$, $j = 1, \ldots, n$. The next slope is obtained by evaluating $p(0)$.

**Using Newton's Method**

Newton's method for solving the nonlinear equation $\phi(z) = 0$ is

$$z_{n+1} = z_n - \frac{\phi(z_n)}{\phi'(z_n)}, \quad n \geq 0.$$

Now the problem is to obtain the value $\phi'(z_n)$. Note that this is anything but obvious, since we do not even have an expression for the function $\phi$ – only for the value $\phi(z_n)$.

In order to obtain an expression for $\phi'(z_n)$ we consider the initial value problem (27) in the form

$$\begin{aligned} x''(t,z) &= f(t, x(t,z), x'(t,z)) \\ x(a,z) &= \alpha, \quad x'(a,z) = z. \end{aligned} \tag{28}$$

We now look at the change of the solution $x$ with respect to the initial slope $z$, i.e.,

$$\begin{aligned} \frac{\partial x''(t,z)}{\partial z} &= \frac{\partial}{\partial z} f(t, x(t,z), x'(t,z)) \\ &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial z} + \frac{\partial f}{\partial x'}\frac{\partial x'}{\partial z}, \end{aligned} \tag{29}$$

where we have omitted the arguments of $f$, $x$, and $x'$ in the second line. The initial conditions become

$$\frac{\partial x}{\partial z}(a,z) = 0, \quad \text{and} \quad \frac{\partial x'}{\partial z}(a,z) = 1.$$

If we introduce the notation $v(t) = \frac{\partial x}{\partial z}(t,z)$, then (29) becomes

$$\begin{gathered} v''(t) = \tfrac{\partial f}{\partial x}(t, x(t), x'(t))v(t) + \tfrac{\partial f}{\partial x'}(t, x(t), x'(t))v'(t) \\ v(a) = 0, \quad v'(a) = 1. \end{gathered} \tag{30}$$

Equation (30) is called the *first variational equation*. We can recognize this as another initial value problem for the function $v$.

Now,

$$\phi(z) = x(b,z) - \beta,$$

so that

$$\phi'(z) = \frac{\partial x}{\partial z}(b,z) = v(b).$$

Therefore, we can obtain the value $\phi'(z_n)$ required in Newton's method by solving the initial value problem (30) up to $t = b$.

**Algorithm**

1. Provide an initial guess $z_0$ and a tolerance $\epsilon$.
2. Solve the initial value problems (27) and (30) with initial conditions

   $$x(a) = \alpha,\ x'(a) = z_0, \quad \text{and} \quad v(a) = 0,\ v'(a) = 1,$$

   respectively. Let $n = 0$. This provides us with $\phi(z_n) = x_{z_n}(b) - \beta$ and $\phi'(z_n) = v(b)$.
3. Apply Newton's method, i.e., compute

   $$z_{n+1} = z_n - \frac{\phi(z_n)}{\phi'(z_n)}.$$

4. Check if $|\phi(z_{n+1})| < \epsilon$. If yes, stop. Otherwise, increment $n$ and repeat from 3.

**Remark:**

1. Again, computing $\phi(z_{n+1})$ in Step 4 requires solution of an IVP (27).
2. The initial value problems (27) and (30) can be solved simultaneously using a vectorized IVP solver.

**Linear Boundary Value Problems**

If the boundary value problem (26) is linear, then the function $\phi$ will also be linear, and therefore a single step of the secant method will provide the "correct" initial slope.

We now illustrate how to implement this special case. Consider the linear BVP

$$\begin{aligned} x''(t) &= u(t) + v(t)x(t) + w(t)x'(t) \\ x(a) &= \alpha, \quad x(b) = \beta. \end{aligned}$$

We need to provide two initial guesses for the associated IVPs (27), i.e., $x'_{z_0}(a) = z_0$ and $x'_{z_1}(a) = z_1$. Now we form a convex combination of the two solutions $x_{z_0}$ and $x_{z_1}$:

$$x(t) = \lambda x_{z_0}(t) + (1-\lambda)x_{z_1}(t).$$

We will determine $\lambda$ so that $x$ satisfies the second boundary condition $x(b) = \beta$. Therefore,

$$\beta = x(b) = \lambda x_{z_0}(b) + (1-\lambda)x_{z_1}(b)$$

or

$$\lambda = \frac{\beta - x_{z_1}(b)}{x_{z_0}(b) - x_{z_1}(b)}.$$

In order to formulate the algorithm for this case we convert the two initial value problems (27) to an autonomous system using the notation $x_0(t) = t$, $x_1(t) = x_{z_0}(t)$, and $x_2(t) = x_{z_1}(t)$.

**Algorithm**

1. Let $z_0 = 0$, $z_1 = 1$, and solve

$$\begin{aligned} x_0'(t) &= 1 \\ x_1'(t) &= x_3(t) \\ x_2'(t) &= x_4(t) \\ x_3'(t) &= u(x_0(t)) + v(x_0(t))x_1(x_0(t)) + w(x_0(t))x_3(t) \\ x_4'(t) &= u(x_0(t)) + v(x_0(t))x_2(x_0(t)) + w(x_0(t))x_4(t) \end{aligned}$$

   with

$$\begin{aligned} x_0(a) &= a \\ x_1(a) &= \alpha \\ x_2(a) &= \alpha \\ x_3(a) &= 0 \\ x_4(a) &= 1. \end{aligned}$$

2. Compute

$$\lambda = \frac{\beta - x_2(b)}{x_1(b) - x_2(b)}.$$

3. To evaluate the solution use

$$x(t) = \lambda x_1(t) + (1-\lambda)x_2(t),$$

   where $x_1$ and $x_2$ were obtained in Step 1.

**Multiple Shooting**

It is also possible to subdivide the interval $[a, b]$, and then apply the shooting method from both ends. This means that additional (internal boundary) conditions need to be formulated that ensure that the solutions match up at the subdivision points. This leads to a system of nonlinear equations which can then be solved with a multivariate Newton or Quasi-Newton method. A few more details are provided in the textbook on page 585–587. A more complete reference is the book "Introduction to Numerical Analysis" by Stoer and Bulirsch (1980).

## 8.10 Boundary Value Problems: Finite Differences

Again we consider the boundary value problem

$$\begin{aligned} x''(t) &= f(t, x(t), x'(t)) \\ x(a) &= \alpha, \quad x(b) = \beta. \end{aligned} \tag{31}$$

Now we create a uniform partition of the interval $[a, b]$ into $n+1$ subintervals $[t_i, t_{i+1}]$, $i = 0, 1, \ldots, n$, where

$$t_i = a + ih, \quad \text{and} \quad h = \frac{b-a}{n+1}.$$

The basic idea is to discretize the differential equation (31) on the given partition. To this end we introduce the following difference approximations to the first and second derivatives involved in (31):

$$\begin{aligned} x'(t) &= \frac{x(t+h) - x(t-h)}{2h} - \frac{h^2}{6} x^{(3)}(\xi_i) \\ x''(t) &= \frac{x(t+h) - 2x(t) + x(t-h)}{h^2} - \frac{h^2}{12} x^{(4)}(\tau_i). \end{aligned} \tag{32}$$

Recall that we introduced these kinds of derivative approximations in Section 7.1. If we use the notation $x_i = x(t_i)$ along with the finite difference approximations (32), then the boundary value problem (31) becomes

$$\begin{aligned} x_0 &= \alpha \\ \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} &= f\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right), \quad i = 1, \ldots, n, \\ x_{n+1} &= \beta. \end{aligned} \tag{33}$$

We now first discuss the case in which $f$ is a *linear* function of $x$ and $x'$, i.e.,

$$f(t, x(t), x'(t)) = u(t) + v(t)x(t) + w(t)x'(t).$$

Then (33) becomes

$$\begin{aligned} x_0 &= \alpha \\ \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} &= u_i + v_i x_i + w_i \frac{x_{i+1} - x_{i-1}}{2h}, \quad i = 1, \ldots, n, \\ x_{n+1} &= \beta, \end{aligned} \tag{34}$$

where we have used the notation $u_i = u(t_i)$, $v_i = v(t_i)$, and $w_i = w(t_i)$. This is a system of $n$ linear equations for the $n$ unknowns $x_i$, $i = 1, \ldots, n$. In fact, the system is *tridiagonal*. This can be seen if we rewrite (34) as

$$\begin{aligned} x_0 &= \alpha \\ \left(-1 - \frac{w_i}{2}h\right) x_{i-1} + \left(2 + h^2 v_i\right) x_i + \left(-1 + \frac{w_i}{2}h\right) x_{i+1} &= -h^2 u_i, \quad i = 1, \ldots, n, \\ x_{n+1} &= \beta, \end{aligned}$$

or in matrix form

$$\begin{bmatrix} 2+h^2v_1 & -1+\frac{w_1}{2}h & 0 & \ldots & 0 \\ -1-\frac{w_2}{2}h & 2+h^2v_2 & -1+\frac{w_2}{2}h & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & -1-\frac{w_{n-1}}{2}h & 2+h^2v_{n-1} & -1+\frac{w_{n-1}}{2}h \\ 0 & \ldots & 0 & -1-\frac{w_n}{2}h & 2+h^2v_n \end{bmatrix} \times$$

$$\times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} -h^2u_1 - \alpha\left(-1-\frac{w_1}{2}h\right) \\ -h^2u_2 \\ \vdots \\ -h^2u_{n-1} \\ -h^2u_n - \beta\left(-1+\frac{w_n}{2}h\right) \end{bmatrix}.$$

**Remark:** As with our earlier solvers for initial value problems (which were also used for the shooting method) the numerical solution is obtained only as a set of discrete values $\{x_i \,:\, i = 0, 1, \ldots, n\}$. However, all values are obtained simultaneously once the linear system is solved.

The tridiagonal system above can be solved most efficiently if we can ensure that it is *diagonally dominant*, since then a tridiagonal Gauss solver without pivoting can be applied. Diagonal dominance for the above system means that we need to ensure

$$|2 + h^2 v_i| > |1 + \frac{h}{2}w_i| + |1 - \frac{h}{2}w_i|.$$

This inequality will be satisfied if we assume $v_i > 0$, and that the discretization is so fine that $|\frac{h}{2}w_i| < 1$. Under these assumptions we get

$$2 + h^2 v_i > 1 + \frac{h}{2}w_i + 1 - \frac{h}{2}w_i = 2 \quad \Longleftrightarrow \quad h^2 v_i > 0$$

which is obviously true.

**Remarks:**

1. The assumption $v_i > 0$ is no real restriction since this is also a condition for the Existence and Uniqueness Theorem 8.15.

2. The assumption $|\frac{h}{2}w_i| < 1$ on the mesh size $h$ is a little more difficult to verify.

## Convergence of the Linear Finite Difference Method

**Theorem 8.16** *The maximum pointwise error of the linear finite difference method is given by*

$$\max_{i=0,1,\ldots,n} |x(t_i) - x_i| \leq Ch^2, \quad \text{as } h \to 0,$$

*where $x(t_i)$ is the exact solution at $t_i$, and $x_i$ is the corresponding approximate solution obtained by the finite difference method.*

**Proof:** For the exact solution we have

$$\frac{x(t_i+h) - 2x(t_i) + x(t_i-h)}{h^2} - \frac{h^2}{12}x^{(4)}(\tau_i) = u_i + v_i x(t_i) + w_i \left[\frac{x(t_i+h) - x(t_i-h)}{2h} - \frac{h^2}{6}x^{(3)}(\xi_i)\right],$$

whereas for the approximate solution we have the relation (34). Subtracting (34) from the above equation yields

$$\frac{e_{i+1} - 2e_i + e_{i-1}}{h^2} = v_i e_i + w_i \frac{e_{i+1} - e_i}{2h} + h^2 g_i, \tag{35}$$

where

$$e_i = x(t_i) - x_i$$

and

$$g_i = \frac{1}{12}x^{(4)}(\tau_i) - \frac{1}{6}x^{(3)}(\xi_i).$$

Since (35) is analogous to (34) it can be rewritten as

$$\left(-1 - \frac{w_i}{2}h\right)e_{i-1} + \left(2 + h^2 v_i\right)e_i + \left(-1 + \frac{w_i}{2}h\right)e_{i+1} = -h^4 g_i.$$

Now we let $\lambda = \|e\|_\infty$, and pick $i$ such that

$$|e_i| = \|e\|_\infty = \lambda.$$

Then we get

$$|2 + h^2 v_i| \underbrace{|e_i|}_{=\lambda} \leq h^4 |g_i| + |-1 + \frac{w_i}{2}h| \underbrace{|e_{i+1}|}_{\leq\lambda} + |-1 - \frac{w_i}{2}h| \underbrace{|e_{i-1}|}_{\leq\lambda}.$$

Using the definition of $\lambda$, and bounding $|g_i|$ by its maximum we have

$$\lambda\left(|2 + h^2 v_i| - |-1 + \frac{w_i}{2}h| - |-1 - \frac{w_i}{2}h|\right) \leq h^4 \|g\|_\infty.$$

Using the same assumptions and arguments as in the diagonal dominance discussion above, the expression in parentheses is equal to $h^2 v_i$, and therefore we have

$$\lambda v_i \leq h^2 \|g\|_\infty,$$

or, since $\lambda = \|e\|_\infty$,

$$\max_{i=0,1,\ldots,n+1} |x(t_i) - x_i| \leq Ch^2,$$

where

$$C = \frac{\|g\|_\infty}{\inf_{a \le t \le b} v(t)}.$$

♠

**Remark:** The error bound in Theorem 8.16 holds only for $C^4$ functions $x$, whereas for the solution to exist only $C^2$ continuity is required.

**Nonlinear Finite Differences**

We now return to the original discretization

$$\begin{aligned} x_0 &= \alpha \\ \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} &= f\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right), \quad i = 1, \ldots, n, \\ x_{n+1} &= \beta \end{aligned}$$

of the boundary value problem (31). However, now we allow $f$ to be a nonlinear function. This leads to the following system of *nonlinear* equations:

$$\begin{aligned} 2x_1 - x_2 + h^2 f\left(t_1, x_1, \frac{x_2 - \alpha}{2h}\right) - \alpha &= 0 \\ -x_{i-1} + 2x_i - x_{i+1} + h^2 f\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right) &= 0, \quad i = 2, \ldots, n-1, \\ -x_{n-1} + 2x_n + h^2 f\left(t_n, x_n, \frac{\beta - x_{n-1}}{2h}\right) - \beta &= 0. \end{aligned} \tag{36}$$

One can show that this system has a unique solution provided

$$h < \frac{2}{M},$$

where $M$ is the same as in the Existence and Uniqueness Theorem 8.14.

To solve the system we need to apply Newton iteration for nonlinear systems. This is done by solving the linear system

$$J(\mathbf{x}^{(k)})\mathbf{u} = -F(\mathbf{x}^{(k)})$$

for $\mathbf{u}$, and then updating

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{u},$$

where $\mathbf{x}^{(k)}$ is the $k$-th iterate of the vector of grid values $x_0, x_1, \ldots, x_{n+1}$, and $J$ is the tridiagonal Jacobian matrix defined by

$$J(\mathbf{x})_{ij} = \begin{cases} -1 + \frac{h}{2} f_{x'}\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right), & i = j-1,\ j = 2, \ldots, n, \\ 2 + h^2 f_x\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right), & i = j,\ j = 1, \ldots, n, \\ -1 - \frac{h}{2} f_{x'}\left(t_i, x_i, \frac{x_{i+1} - x_{i-1}}{2h}\right), & i = j+1,\ j = 1, \ldots, n-1. \end{cases}$$

$F(\mathbf{x})$ is given by the left-hand side of the equations in (36).

**Remarks:**

1. As always, Newton iteration requires a "good" initial guess $x_1, \ldots, x_n$.
2. One can show that the nonlinear finite difference method also has $\mathcal{O}(h^2)$ convergence order.

## 8.11 Boundary Value Problems: Collocation

We now present a different type of numerical method that will yield the approximate solution of a two-point boundary value problem in the form of a function, as opposed to the set of discrete points resulting from the shooting or finite difference methods.

Assume we are given a general linear boundary value problem of the form

$$\begin{aligned} Lx(t) &= f(t), \qquad t \in [a,b], \\ x(a) &= \alpha, \quad x(b) = \beta. \end{aligned} \tag{37}$$

To keep the discussion as general as possible, we now let

$$V = \text{span}\{v_1, \ldots, v_n\}$$

denote an approximation space we wish to represent the approximate solution in. We can think of $V$ as being, e.g., the space of polynomials or splines of a certain degree, or some radial basis function space. We will express the approximate solution in the form

$$x(t) = \sum_{j=1}^{n} c_j v_j(t), \qquad t \in [a,b],$$

with unknown coefficients $c_1, \ldots, c_n$. Since $L$ is assumed to be linear we have

$$Lx = \sum_{j=1}^{n} c_j L v_j,$$

and (37) becomes

$$\begin{aligned} &\sum_{j=1}^{n} c_j L v_j(t) = f(t), \qquad t \in [a,b], \\ &\sum_{j=1}^{n} c_j v_j(a) = \alpha, \quad \sum_{j=1}^{n} c_j v_j(b) = \beta. \end{aligned} \tag{38}$$

In order to determine the $n$ unknown coefficients $c_1, \ldots, c_n$ in this formulation we impose $n$ collocation conditions to obtain an $n \times n$ system of linear equations for the $c_j$.

The last two equations in (38) ensure that the boundary conditions are satisfied, and give us the first two collocation equations. To obtain the other $n-2$ equations we choose $n-2$ *collocation points* $\tau_2, \ldots, \tau_{n-1}$, at which we enforce the differential equation. As in the previous numerical methods, this results in a discretization of the differential equation.

If we let $\tau_1 = a$ and $\tau_n = b$, then (38) becomes

$$\begin{aligned} \sum_{j=1}^{n} c_j v_j(\tau_1) &= \alpha, \\ \sum_{j=1}^{n} c_j L v_j(\tau_i) &= f(\tau_i), \qquad i = 2, \ldots, n-1, \end{aligned}$$

$$\sum_{j=1}^{n} c_j v_j(\tau_n) \quad = \quad \beta.$$

In matrix form we have the linear system

$$\begin{bmatrix} v_1(\tau_1) & v_2(\tau_1) & \dots & v_n(\tau_1) \\ Lv_1(\tau_2) & Lv_2(\tau_2) & \dots & Lv_n(\tau_2) \\ \vdots & & & \vdots \\ Lv_1(\tau_{n-1}) & Lv_2(\tau_{n-1}) & \dots & Lv_n(\tau_{n-1}) \\ v_1(\tau_n) & v_2(\tau_n) & \dots & v_n(\tau_n) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \alpha \\ f(\tau_2) \\ \vdots \\ f(\tau_{n-1}) \\ \beta \end{bmatrix}. \tag{39}$$

If the space $V$ and the collocation points $\tau_i$, $i = 1, \dots, n$, are chosen such that the collocation matrix in (39) is nonsingular then we can represent an approximate solution of (37) from the space $V$ uniquely as

$$x(t) = \sum_{j=1}^{n} c_j v_j(t), \qquad t \in [a, b].$$

**Remark:** Note that this provides the solution in the form of a function that can be evaluated anywhere in $[a, b]$. No additional interpolation is required as was the case with the earlier methods.

**Radial Basis Functions for Collocation**

The following discussion will be valid for any sufficiently smooth radial basic function. However, to be specific, we will choose the multiquadric basic function

$$\phi(r) = \sqrt{r^2 + \sigma^2}, \quad \sigma > 0,$$

with $r = |\cdot - \tau|$. If we center a multiquadric at each one of the collocation points $\tau_j$, $j = 1, \dots, n$, then the approximation space becomes

$$V = \text{span}\{\phi(|\cdot - \tau_j|),\ j = 1, \dots, n\}.$$

Now the system (39) becomes

$$\begin{bmatrix} \phi(|\tau_1 - \tau_1|) & \phi(|\tau_1 - \tau_2|) & \dots & \phi(|\tau_1 - \tau_n|) \\ L\phi(|\tau_2 - \tau_1|) & L\phi(|\tau_2 - \tau_2|) & \dots & L\phi(|\tau_2 - \tau_n|) \\ \vdots & & & \vdots \\ L\phi(|\tau_{n-1} - \tau_1|) & L\phi(|\tau_{n-1} - \tau_2|) & \dots & L\phi(|\tau_{n-1} - \tau_n|) \\ \phi(|\tau_n - \tau_1|) & \phi(|\tau_n - \tau_2|) & \dots & \phi(|\tau_n - \tau_n|) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \alpha \\ f(\tau_2) \\ \vdots \\ f(\tau_{n-1}) \\ \beta \end{bmatrix}. \tag{40}$$

To get a better feel for this system we consider an example.

**Example:** Let the differential operator $L$ be given by

$$Lx(t) = x''(t) + \lambda x'(t) + \mu x(t),$$

and $\phi$ denote the multiquadric radial basic function. Then

$$L\phi(|t - \tau|) = \phi''(|t - \tau|) + \lambda \phi'(|t - \tau|) + \mu \phi(|t - \tau|)$$

with

$$\begin{aligned}\phi'(|t-\tau|) &= \frac{d}{dt}\phi(|t-\tau|)\\ &= \frac{d}{dt}\sqrt{|t-\tau|^2+\sigma^2}\\ &= \frac{t-\tau}{\sqrt{|t-\tau|^2+\sigma^2}}\end{aligned}$$

and

$$\begin{aligned}\phi''(|t-\tau|) &= \frac{d}{dt}\phi'(|t-\tau|)\\ &= \frac{d}{dt}\frac{t-\tau}{\sqrt{|t-\tau|^2+\sigma^2}}\\ &= \frac{\sqrt{|t-\tau|^2+\sigma^2}-\frac{(t-\tau)^2}{\sqrt{|t-\tau|^2+\sigma^2}}}{|t-\tau|^2+\sigma^2}\\ &= \frac{\sigma^2}{(|t-\tau|^2+\sigma^2)^{3/2}}.\end{aligned}$$

Therefore, we get

$$L\phi(|t-\tau|) = \frac{\sigma^2}{(|t-\tau|^2+\sigma^2)^{3/2}} + \lambda\frac{t-\tau}{\sqrt{|t-\tau|^2+\sigma^2}} + \mu\sqrt{|t-\tau|^2+\sigma^2},$$

and the collocation matrix has entries of this type in rows 2 through $n-1$.

**Remarks:**

1. This method was suggested by Kansa (1990) and is one of the most popular approaches for solving boundary value problems with radial basis functions. The popularity of this method is due to the fact that it is simple to implement and it generalizes in a straightforward way to boundary value problems for elliptic partial differential equations in higher space dimensions.

2. It was not known for a long time whether the matrix for this kind of radial basis function collocation was nonsingular. However, recently Hon and Schaback (2001) showed that there exist configurations of collcoation points (in the elliptic PDE setting in $\mathbb{R}^2$) for which the matrix will be singular for many of the most popular radial basis functions.

It is obvious that the matrix in (40) is not symmetric. This means that many efficient linear algebra subroutines cannot be employed in its solution. Another approach to radial basis function collocation which yields a symmetric matrix for operators $L$ with even order derivatives was suggested by Fasshauer (1997).

We now use a different approximation space, namely

$$V = \text{span}\{\phi(|\cdot-\tau_1|), \phi(|\cdot-\tau_n|)\} \cup \text{span}\{L^{(2)}\phi(|\cdot-\tau_j|),\ j=2,\ldots,n-1\}.$$

Here the operator $L^{(2)}$ is identical to $L$, but acts on $\phi$ as a function of the second variable $\tau$.

Since the approximate solution is now of the form

$$x(t) = c_1\phi(|t-\tau_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|t-\tau_j|) + c_n\phi(|t-\tau_n|) \tag{41}$$

we need to look at the collocation system one more time.

We start with (38), which – based on (41) – now becomes

$$\begin{aligned}
c_1\phi(|a-\tau_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|a-\tau_j|) + c_n\phi(|a-\tau_n|) &= \alpha, \\
c_1 L\phi(|t-\tau_1|) + \sum_{j=2}^{n-1} c_j L L^{(2)}\phi(|t-\tau_j|) + c_n L\phi(|t-\tau_n|) &= f(t), \qquad t \in [a,b], \\
c_1\phi(|b-\tau_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|b-\tau_j|) + c_n\phi(|b-\tau_n|) &= \beta.
\end{aligned}$$

If we enforce the collocation conditions at the interior points $\tau_2, \ldots, \tau_{n-1}$, then we get the system of linear equations

$$\begin{bmatrix}
\phi(|a-\tau_1|) & L^{(2)}\phi(|a-\tau_2|) & \ldots & L^{(2)}\phi(|a-\tau_{n-1}|) & \phi(|a-\tau_n|) \\
L\phi(|\tau_2-\tau_1|) & LL^{(2)}\phi(|\tau_2-\tau_2|) & \ldots & LL^{(2)}\phi(|\tau_2-\tau_{n-1}|) & L\phi(|\tau_2-\tau_n|) \\
\vdots & & & \vdots & \\
L\phi(|\tau_{n-1}-\tau_1|) & LL^{(2)}\phi(|\tau_{n-1}-\tau_2|) & \ldots & LL^{(2)}\phi(|\tau_{n-1}-\tau_{n-1}|) & L\phi(|\tau_{n-1}-\tau_n|) \\
\phi(|b-\tau_1|) & L^{(2)}\phi(|b-\tau_2|) & \ldots & L^{(2)}\phi(|b-\tau_{n-1}|) & \phi(|b-\tau_n|)
\end{bmatrix} \times$$

$$\times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} \alpha \\ f(\tau_2) \\ \vdots \\ f(\tau_{n-1}) \\ \beta \end{bmatrix}. \tag{42}$$

**Remarks:**

1. If $L = L^{(2)}$, i.e., only even-order derivatives are used (see the example below), then the matrix in (42) is symmetric as claimed earlier.

2. Depending on whether globally or locally supported radial basis functions are being used, we can now employ efficient linear solvers, such as Cholesky factorization or the conjugate gradient method, to solve this system.

3. The most important advantage of the symmetric collocation method over the non-symmetric one proposed by Kansa is that one can prove that the collocation matrix in the symmetric case is nonsingular for all of the standard radial basis functions and any choice of distinct collocation points.

4. Another benefit of using the symmetric form is that it is possible to give convergence order estimates for this case.

5. Since terms of the form $LL^{(2)}\phi$ are used, the symmetric collocation method has the disadvantage that it requires higher smoothness. Moreover, computing and coding these terms is more complicated than for the non-symmetric collocation method.

**Example:** We again consider the differential operator $L$ given by

$$Lx(t) = x''(t) + \lambda x'(t) + \mu x(t),$$

and multiquadrics. Then

$$\begin{aligned} L^{(2)}\phi(|t-\tau|) &= \frac{d^2}{d\tau^2}\phi(|t-\tau|) + \lambda\frac{d}{d\tau}\phi(|t-\tau|) + \mu\phi(|t-\tau|) \\ &= \frac{\sigma^2}{(|t-\tau|^2+\sigma^2)^{3/2}} - \lambda\frac{t-\tau}{\sqrt{|t-\tau|^2+\sigma^2}} + \mu\sqrt{|t-\tau|^2+\sigma^2}, \end{aligned}$$

which is almost the same as $L\phi(|t-\tau|)$ above except for the sign difference in the first derivative term. The higher-order terms are rather complicated. In the special case $\lambda = \mu = 0$ we get

$$LL^{(2)}\phi(|t-\tau|) = \frac{15(t-\tau)^2\sigma^2}{(|t-\tau|^2+\sigma^2)^{7/2}} - \frac{3\sigma^2}{(|t-\tau|^2+\sigma^2)^{5/2}} + \frac{\sigma^2}{(|t-\tau|^2+\sigma^2)^{3/2}}.$$

**Remark:** Collocation with cubic splines is described in the textbook on pages 595–596.