1 Ordinary Differential Equations

1.0 Mathematical Background

1.0.1 Smoothness

Definition 1.1 *A function f defined on $[a,b]$ is* continuous at $\xi \in [a,b]$ *if* $\lim_{x\to\xi} f(x) = f(\xi)$.

Remark Note that this implies existence of the quantities on both sides of the equation.

f is *continuous on* $[a,b]$, i.e., $f \in C[a,b]$, if f is continuous at every $\xi \in [a,b]$.
If $f^{(\nu)}$ is continuous on $[a,b]$, then $f \in C^{(\nu)}[a,b]$.
Alternatively, one could start with the following ε-δ definition:

Definition 1.2 *A function f defined on $[a,b]$ is continuous at $\xi \in [a,b]$ if for every $\varepsilon > 0$ there exists a $\delta_\varepsilon > 0$ (that depends on ξ) such that $|f(x) - f(\xi)| < \varepsilon$ whenever $|x - \xi| < \delta_\varepsilon$.*

FIGURE

Example (A function that is continuous, but not uniformly) $f(x) = \frac{1}{x}$ with FIGURE.

Definition 1.3 *A function f is* uniformly continuous on $[a,b]$ *if it is continuous with a uniform δ_ε for all $\xi \in [a,b]$, i.e., independent of ξ.*

Important for ordinary differential equations is

Definition 1.4 *A function f defined on $[a,b]$ is* Lipschitz continuous on $[a,b]$ *if there exists a number λ such that $|f(x) - f(y)| \le \lambda |x - y|$ for all $x, y \in [a,b]$. λ is called the* Lipschitz constant.

Remark

1. In fact, any Lipschitz continuous function is *uniformly* continuous, and therefore continuous.
2. For a differentiable function with bounded derivative we can take
$$\lambda = \max_{\xi\in[a,b]} |f'(\xi)|,$$
and we see that Lipschitz continuity is "between" continuity and differentiability.
3. If the function f is Lipschitz continuous on $[a,b]$ with Lipschitz constant λ, then f is almost everywhere differentiable in $[a,b]$ with $|f'(x)| \le \lambda$. In other words, Lipschitz continuous functions need not be differentiable everywhere in $[a,b]$.
4. See also Assignment 1.

1.0.2 Polynomials

A *polynomial of degree at most* ν is of the form

$$p(x) = \sum_{k=0}^{\nu} p_k x^k$$

with $p_k, x \in \mathbb{R}$. Notation: $p \in \mathbb{P}_\nu$.

Theorem 1.5 *(Taylor's Theorem) If* $f \in C^\nu[a,b]$ *and* $f^{(\nu+1)}$ *exists on* (a,b)*, then for any* $x \in [a,b]$

$$f(x) = \underbrace{\sum_{k=0}^{\nu} \frac{1}{k!} f^{(k)}(\xi)(x-\xi)^k}_{\textit{Taylor polynomial}} + \underbrace{\frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta)(x-\xi)^{\nu+1}}_{\textit{error term}},$$

where η *is some point between* x *and* ξ*.*

FIGURE

An alternate form commonly used is

$$f(\xi + h) = \sum_{k=0}^{\nu} \frac{1}{k!} f^{(k)}(\xi) h^k + \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta) h^{\nu+1}.$$

Remark Note that the information about f is provided locally, at ξ only.

If the information is spread out, i.e., when we are given distinct points $\xi_0 < \xi_1 < \ldots < \xi_\nu \in [a,b]$ and associated values $f(\xi_0), f(\xi_1), \ldots, f(\xi_\nu)$, then there exists a *unique interpolation polynomial*

$$p(x) = \sum_{k=0}^{\nu} p_k(x) f(\xi_k)$$

with

$$p_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{\nu} \frac{x - \xi_j}{\xi_k - \xi_j}, \qquad k = 0, 1, \ldots, \nu,$$

such that $p(\xi_k) = f(\xi_k)$, $k = 0, 1, \ldots, \nu$. The p_k are called *Lagrange functions*, and the polynomial is said to be in *Lagrange form*.

Example We now compute the Lagrange form of the polynomial interpolating the data (with FIGURE)

ξ	0	1	3
$f(\xi)$	1	0	4

.

We have

$$p(x) = \sum_{k=0}^{2} p_k(x) f(\xi_k) = p_0(x) + 4p_2(x),$$

where

$$p_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{2} \frac{x - \xi_j}{\xi_k - \xi_j}.$$

Thus,

$$p_0(x) = \frac{(x-\xi_1)(x-\xi_2)}{(\xi_0-\xi_1)(\xi_0-\xi_2)} = \frac{(x-1)(x-3)}{(-1)(-3)} = \frac{1}{3}(x-1)(x-3),$$

and

$$p_2(x) = \frac{(x-\xi_0)(x-\xi_1)}{(\xi_2-\xi_0)(\xi_2-\xi_1)} = \frac{x(x-1)}{(3-0)(3-1)} = \frac{1}{6}x(x-1).$$

This gives us

$$p(x) = \frac{1}{3}(x-1)(x-3) + \frac{2}{3}x(x-1) = (x-1)^2.$$

Theorem 1.6 *(Polynomial interpolation error) For every $x \in [a,b]$ there exists an $\eta = \eta(x) \in [a,b]$ such that*

$$p(x) - f(x) = \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta) \prod_{k=0}^{\nu} (x - \xi_k), \tag{1}$$

where p is the degree ν interpolating polynomial to f, i.e., $p(\xi_k) = f(\xi_k)$, $k = 0, 1, \ldots, \nu$.

Remark See Assignment 1 for an illustration of Taylor vs. interpolation polynomials.

In order to prove this result we need to recall Rolle's Theorem:

Theorem 1.7 *If $f \in C[a,b]$ and f' exists on (a,b), and if $f(a) = f(b) = 0$, then there exists a number $\eta \in (a,b)$ such that $f'(\eta) = 0$.*

Proof (of Theorem 1.6) If x coincides with one of the data sites ξ_k, $k = 0, 1, \ldots, \nu$, then it is easy to see that both sides of equation (1) are zero.

Thus we now assume $x \neq \xi_k$ to be fixed. We start be defining

$$w(t) = \prod_{k=0}^{\nu} (t - \xi_k)$$

and

$$F = f - p - \alpha w$$

with α such that $F(x) = 0$, i.e.,

$$\alpha = \frac{f(x) - p(x)}{w(x)}.$$

We need to show that $\alpha = \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta)$ for some $\eta \in (a,b)$.

Since $f \in C^{\nu+1}[a,b]$ we know that $F \in C^{\nu+1}[a,b]$ also. Moreover,

$$F(t) = 0 \quad \text{for} \quad t = x, \xi_0, \xi_1, \ldots, \xi_\nu.$$

The first of these equations holds by the definition of α, the remainder by the definition of w and the fact that p interpolates f at these points.

Now we apply Rolle's Theorem to F on each of the $\nu+1$ subintervals generated by the $\nu+2$ points $x, \xi_0, \xi_1, \ldots, \xi_\nu$. Thus, F' has (at least) $\nu+1$ distinct zeros in (a,b).

Next, by Rolle's Theorem (applied to F' on ν subintervals) we know that F'' has (at least) ν zeros in (a,b).

Continuing this argument we deduce that $F^{(\nu+1)}$ has (at least) one zero, η, in (a,b).

On the other hand, since

$$F(t) = f(t) - p(t) - \alpha w(t)$$

we have

$$F^{(\nu+1)}(t) = f^{(\nu+1)}(t) - p^{(\nu+1)}(t) - \alpha w^{(\nu+1)}(t).$$

However, p is a polynomial of degree at most ν, so $p^{(\nu+1)} \equiv 0$. Since the leading coefficient of the $\nu+1$st degree polynomial w is 1 we have

$$w^{(\nu+1)}(t) = \frac{d^{\nu+1}}{dt^{\nu+1}} \prod_{k=0}^{\nu} (t - \xi_k) = (\nu+1)!.$$

Therefore,

$$F^{(\nu+1)}(t) = f^{(\nu+1)}(t) - \alpha(\nu+1)!.$$

Combining this with the information about the zero of $F^{(\nu+1)}$ we get

$$\begin{aligned} 0 \;=\; F^{(\nu+1)}(\eta) &= f^{(\nu+1)}(\eta) - \alpha(\nu+1)! \\ &= f^{(\nu+1)}(\eta) - \frac{f(x)-p(x)}{w(x)}(\nu+1)! \end{aligned}$$

or

$$f(x) - p(x) = f^{(\nu+1)}(\eta)\frac{w(x)}{(\nu+1)!}. \quad \blacksquare$$

1.0.3 Peano Kernels

A useful (and rather beautiful) tool for error estimates (especially for numerical differentiation and integration problems) is the use of *Peano kernels* and the Peano kernel theorem.

A *linear functional* L on a linear space, e.g., $C^\nu[a,b]$, is a mapping that maps a function from this space onto a scalar.

Example 1. Point evaluation functional:

$$Lf = f(x).$$

2. (Definite) Integration functional:

$$Lf = \int_a^b f(x)dx.$$

Note that linear combinations of linear functionals form another linear functional. A fairly general linear functional is

$$Lf = \sum_{i=0}^{n} \left[\int_a^b \alpha_i(x) f^{(i)}(x) dx + \sum_{j=1}^{n} \beta_{ij} f^{(i)}(\xi_{ij}) \right]. \tag{2}$$

Here $\xi_{ij} \in [a,b]$, and the functions α_i and β_{ij} are at least piecewise continuous on $[a,b]$. The function f should be in $C^n[a,b]$.

Furthermore, we say that a functional *annihilates polynomials* $\mathbb{P}_\nu$ if

$$Lp = 0, \qquad \text{for all } p \in \mathbb{P}_\nu.$$

The *Peano kernel* of L as in (2) is the function

$$k_\nu(\xi) = L\left[(x-\xi)_+^\nu\right], \qquad \xi \in [a,b],$$

where $\nu \geq n$ and

$$(x-\xi)_+^m = \begin{cases} (x-\xi)^m, & x \geq \xi \\ 0, & x < \xi, \end{cases}$$

is the *truncated power function.*

Theorem 1.8 *(Peano Kernel Theorem) If a functional L of the form (2) annihilates polynomials $\mathbb{P}_\nu$, then for all $f \in C^{\nu+1}[a,b]$,*

$$Lf = \frac{1}{\nu!} \int_a^b k_\nu(\xi) f^{(\nu+1)}(\xi) d\xi$$

where $\nu \geq n$ and k_ν is the Peano kernel of L.

Remark The Peano kernel theorem allows estimates of the form

$$\begin{aligned} |Lf| &\leq \frac{1}{\nu!} \|k_\nu\|_1 \|f^{(\nu+1)}\|_\infty, \\ |Lf| &\leq \frac{1}{\nu!} \|k_\nu\|_\infty \|f^{(\nu+1)}\|_1, \\ |Lf| &\leq \frac{1}{\nu!} \|k_\nu\|_2 \|f^{(\nu+1)}\|_2, \end{aligned}$$

where we used the *norms*

$$\begin{aligned} \|f\|_1 &= \int_a^b |f(x)| dx, \\ \|f\|_2 &= \left(\int_a^b |f(x)|^2 dx \right)^{1/2}, \\ \|f\|_\infty &= \max_{x \in [a,b]} |f(x)|. \end{aligned}$$

Example Consider the integral

$$\int_0^1 f(\xi)d\xi$$

and find an approximate integration formula of the form

$$\int_0^1 f(\xi)d\xi \approx b_1 f(0) + b_2 f(\frac{1}{2}) + b_3 f(1)$$

that is exact if f is a polynomial in $\mathbb{P}_3$, and find its error.

To answer this question we consider the linear functional

$$Lf = \int_0^1 f(\xi)d\xi - b_1 f(0) + b_2 f(\frac{1}{2}) + b_3 f(1),$$

and first find b_1, b_2, and b_3 so that L annihilates $\mathbb{P}_3$.

If we let $f(x) = 1$, then we get the condition

$$0 = Lf = \int_0^1 1d\xi - (b_1 + b_2 + b_3) = 1 - b_1 - b_2 - b_3.$$

For $f(x) = x$ we get

$$0 = Lf = \int_0^1 \xi d\xi - (\frac{1}{2}b_2 + b_3) = \frac{1}{2} - \frac{1}{2}b_2 - b_3,$$

for $f(x) = x^2$ we get

$$0 = Lf = \int_0^1 \xi^2 d\xi - (\frac{1}{4}b_2 + b_3) = \frac{1}{3} - \frac{1}{4}b_2 - b_3,$$

and for $f(x) = x^3$ we get

$$0 = Lf = \int_0^1 \xi^3 d\xi - (\frac{1}{8}b_2 + b_3) = \frac{1}{4} - \frac{1}{8}b_2 - b_3.$$

The unique solution of this system of 4 linear equations in 3 unknowns is

$$b_1 = \frac{1}{6},\ b_2 = \frac{2}{3},\ b_3 = \frac{1}{6},$$

and therefore

$$\int_0^1 f(\xi)d\xi \approx \frac{1}{6}\left[f(0) + 4f(\frac{1}{2}) + f(1)\right].$$

To estimate the error in this approximation we use the Peano kernel of L. It is given by

$$\begin{aligned}
k_3(\xi) &= L\left[(x-\xi)_+^3\right] \\
&= \int_0^1 (x-\xi)_+^3 dx - \frac{1}{6}\left[(0-\xi)_+^3 + 4(\frac{1}{2}-\xi)_+^3 + (1-\xi)_+^3\right] \\
&= \int_\xi^1 (x-\xi)^3 dx - \frac{1}{6}\left[4(\frac{1}{2}-\xi)_+^3 + (1-\xi)_+^3\right] \\
&= \frac{(1-\xi)^4}{4} - \frac{1}{6}\begin{cases}\left[4(\frac{1}{2}-\xi)^3 + (1-\xi)^3\right], & 0 \le \xi \le \frac{1}{2} \\ (1-\xi)^3, & \frac{1}{2} \le \xi \le 1.\end{cases} \\
&= \begin{cases}-\frac{1}{12}\xi^3(2-3\xi), & 0 \le \xi \le \frac{1}{2} \\ -\frac{1}{12}(1-\xi)^3(3\xi-1), & \frac{1}{2} \le \xi \le 1.\end{cases}
\end{aligned}$$

Now the Peano kernel theorem says that

$$\int_0^1 f(\xi)d\xi - \frac{1}{6}\left[f(0) + 4f(\frac{1}{2}) + f(1)\right] = Lf = \frac{1}{3!}\int_0^1 k_3(\xi)f^{(4)}(\xi)d\xi,$$

and we can explicitly calculate estimates of the form

$$|Lf| \le \frac{1}{1152}\|f^{(4)}\|_1,\ |Lf| \le \frac{\sqrt{14}}{8064}\|f^{(4)}\|_2,\ |Lf| \le \frac{1}{2880}\|f^{(4)}\|_\infty \qquad f \in C^4[0,1]$$

since

$$\|k_3\|_1 = \frac{1}{480}, \quad \|k_3\|_2 = \frac{\sqrt{14}}{1344}, \quad \|k_3\|_\infty = \frac{1}{192}.$$

1.1 ODEs and the Lipschitz Condition

We consider the *system of first-order ODE IVP*

$$\begin{aligned} \boldsymbol{y}'(t) &= \frac{d\boldsymbol{y}(t)}{dt} = \boldsymbol{f}(t, \boldsymbol{y}(t)), \qquad t \ge t_0, & (3)\\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0. & (4) \end{aligned}$$

Here

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_d \end{bmatrix}, \quad \boldsymbol{y}_0 = \begin{bmatrix} y_{0,1} \\ \vdots \\ y_{0,d} \end{bmatrix}, \quad \boldsymbol{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_d \end{bmatrix}, \quad \in \mathbb{R}^d.$$

Remark This approach covers not only first-order ODEs, but also higher-order ODEs, since any d-th order ODE IVP can be converted to a system of d first-order IVPs (see Assignment 1).

If $\boldsymbol{f}(t, \boldsymbol{y}) = A(t)\boldsymbol{y} + \boldsymbol{b}(t)$ for some $d \times d$ matrix-valued function A and $d \times 1$ vector-valued function $\boldsymbol{b}$, then the ODE is *linear*, and if $\boldsymbol{b}(t) = \boldsymbol{0}$ it is linear and *homogeneous*. Otherwise it is *nonlinear*. If $\boldsymbol{f}$ is independent of t, the ODE is called *autonomous*, and if $\boldsymbol{f}$ is independent of $\boldsymbol{y}$, then the ODE system reduces to a (vector of) indefinite integral(s).

Theorem 1.9 *(Picard-Lindelöf: Existence and Uniqueness) Let $\mathbb{B}$ be the ball $\mathbb{B} = \{\boldsymbol{x} \in \mathbb{R}^d :\ \|\boldsymbol{x} - \boldsymbol{y}_0\| \le b\}$ and let $\mathbb{S}$ be the cylinder*

$$\mathbb{S} = \{(t, \boldsymbol{x}) :\ t \in [t_0, t_0 + a],\ \boldsymbol{x} \in \mathbb{B}\}$$

where $a, b > 0$. If $\boldsymbol{f}$ is continuous on $\mathbb{S}$ and $\boldsymbol{f}$ also satisfies the Lipschitz condition

$$\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \le \lambda\|\boldsymbol{x} - \boldsymbol{y}\|, \qquad \boldsymbol{x}, \boldsymbol{y} \in \mathbb{B},$$

then the IVP (24), (4) has a unique solution on $[t_0, t_0 + \alpha]$, where α is some constant that depends on a, b and $\boldsymbol{f}$. In fact,

$$\alpha = \min\left\{a, \frac{b}{\sup_{(t,\boldsymbol{x})\in\mathbb{S}} \|\boldsymbol{f}(t, \boldsymbol{x})\|}\right\}.$$

Note that in the system setting we need to measure differences of vectors in some appropriate *norm* instead of simple absolute value.

Remark The proof of this theorem is rather involved.

Example For a single equation, continuity of the partial derivative $\frac{\partial f(t,y)}{\partial y}$ on $\mathbb{S}$ guarantees Lipschitz continuity of f with

$$\lambda = \max_{\substack{t\in[t_0,t_0+a]\\ y\in\mathbb{B}}} \left|\frac{\partial f(t,y)}{\partial y}\right|.$$

For the initial value problem

$$y'(t) = 2t\,(y(t))^2\,, \qquad y(0) = 1,$$

we have

$$f(t,y) = 2ty^2, \qquad \frac{\partial f(t,y)}{\partial y} = 4ty,$$

which are both continuous on all of $\mathbb{R}^2$. The theorem above guarantees existence and uniqueness of a solution for t near $t_0 = 0$. In fact, it is given by

$$y(t) = \frac{1}{1-t^2}, \qquad -1 < t < 1.$$

However, we see that just because f and $\frac{\partial f(t,y)}{\partial y}$ are continuous on all of $\mathbb{R}^2$ we cannot expect existence or uniqueness of a solution y for all t.

Remark In the system setting a sufficient condition for Lipschitz continuity of $\boldsymbol{f}$ is given by continuity of the *Jacobian matrix*

$$\frac{\partial \boldsymbol{f}(t,\boldsymbol{y})}{\partial \boldsymbol{y}} = \left[\frac{\partial f_i(t,y_1,\ldots,y_d)}{\partial y_j}\right]_{i,j=1}^{d}.$$

Remark Recall that a linear system

$$\boldsymbol{y}' = A\boldsymbol{y}, \qquad t \geq t_0, \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0$$

with $d \times d$ matrix A always has a unique solution. It is given by

$$\boldsymbol{y}(t) = \sum_{\ell=1}^{d} e^{\lambda_\ell (t-t_0)}\boldsymbol{\alpha}_\ell, \qquad t \geq t_0,$$

where $\lambda_1,\ldots,\lambda_d$ are the eigenvalues of A, and the $\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_d \in \mathbb{R}^d$ are vectors (eigenvectors if the eigenvalues are distinct).

1.2 Euler's Method

1.2.1 The basic Algorithm

Recall that we are interested in general first-order IVPs (24), (4) of the form

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)), \qquad t \geq t_0 \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0. \end{aligned}$$

It is our goal to derive numerical methods for the solution of this kind of problem. The first, and probably best known, method is called *Euler's method*. Even though this is one of the "original" numerical methods for the solution of IVPs, it remains important for both practical and theoretical purposes.

The method is derived by considering the approximation

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

of the first derivative. This implies

$$\boldsymbol{y}(t+h) \approx \boldsymbol{y}(t) + h\boldsymbol{y}'(t),$$

which – using the differential equation (24) – becomes

$$\boldsymbol{y}(t+h) \approx \boldsymbol{y}(t) + h\boldsymbol{f}(t, \boldsymbol{y}(t)). \tag{5}$$

Introducing a sequence of points $t_0, t_1 = t_0 + h, t_2 = t_0 + 2h, \ldots, t_N = t_0 + Nh$, this immediately leads to an iterative algorithm.

Algorithm

Input t_0, $\boldsymbol{y}_0$, $\boldsymbol{f}$, h, N

$t = t_0$, $\boldsymbol{y} = \boldsymbol{y}_0$

for $n = 1$ to N do

$\boldsymbol{y} \leftarrow \boldsymbol{y} + h\boldsymbol{f}(t, \boldsymbol{y})$
$t \leftarrow t + h$

end

Remark 1. Alternately, we can derive the formula for Euler's method via integration. Since the IVP gives us both an initial condition as well as the slope $\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y})$ of the solution, we can assume that the slope is constant on a small interval $[t_0, t_0 + h]$, i.e., $\boldsymbol{f}(t, \boldsymbol{y}(t)) \approx \boldsymbol{f}(t_0, \boldsymbol{y}(t_0))$ for $t \in [t_0, t_0 + h]$. Then we can integrate to get

$$\begin{aligned} \boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau \\ &\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(t_0, \boldsymbol{y}(t_0))d\tau \\ &= \boldsymbol{y}(t_0) + (t - t_0)\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) \end{aligned}$$

– Euler's method.

2. Note that Euler's method yields a set of discrete points $(t_n, \boldsymbol{y}_n)$, $n = 1, \ldots, N$, which approximate the graph of the solution $\boldsymbol{y} = \boldsymbol{y}(t)$. In order to obtain a continuous solution one must use an interpolation or approximation method.

3. Euler's method is illustrated in the Maple worksheet `472_Euler_Taylor.mws`.

4. In principle, it is easy to use Euler's method with a variable step size, i.e.,

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h_n \boldsymbol{f}(t_n, \boldsymbol{y}_n),$$

but analysis of the method is simpler with a constant step size $h_n = h$.

1.2.2 Taylor Series Methods

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

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

and therefore obtain the numerical approximation

$$\boldsymbol{y}(t+h) \approx \sum_{k=0}^{\nu} \frac{h^k \boldsymbol{y}^{(k)}(t)}{k!} \tag{6}$$

which is referred to as a ν-th order Taylor series method.

Remark 1. Obviously, Euler's method is a first-order Taylor method.

2. In order to program a Taylor method we need to pre-compute all higher-order derivatives of $\boldsymbol{y}$ required by the method since the differential equation only provides a representation for $\boldsymbol{y}'$. This implies that we will end up with code that depends on (and 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 ($d = 1$)

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

The second-order Taylor approximation is given by

$$y(t+h) \approx y(t) + hy'(t) + \frac{h^2}{2}y''(t).$$

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

From the differential equation

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

Therefore, differentiating this relation,

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

and this can be incorporated into the following algorithm.

Algorithm

Input t_0, y_0, f, h, N

$t = t_0$, $y = y_0$

for $n = 1$ to N do

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

end

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

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

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

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

2. An example of a fourth-order Taylor method is given in the Maple worksheet `472_Euler_Taylor.mws`.

1.2.3 Errors and Convergence

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

$$E_\nu = \frac{h^{\nu+1}}{(\nu+1)!}y^{(\nu+1)}(t + \theta h), \qquad 0 < \theta < 1.$$

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

The global truncation error is the error that results if we use a ν-th order Taylor method having $\mathcal{O}(h^{\nu+1})$ local truncation error to solve our IVP up to time $t = t_0 + t^*$. Since we will be performing

$$N = \left\lfloor \frac{t^*}{h} \right\rfloor$$

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 $\mathcal{O}(h^\nu)$.

Remark

- Of course, truncation errors are independent of roundoff errors which can add to the overall error.
- As we will see later, a method with $\mathcal{O}(h^{\nu+1})$ local accuracy need not be globally ν-th order. In fact, it need not converge at all. *Stability* will be the key to convergence.

A numerical method for the IVP (24), (4) is called *convergent* if for every Lipschitz function $\boldsymbol{f}$ and every $t^* > 0$ we have

$$\lim_{h \to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{y}_{n,h} - \boldsymbol{y}(t_n)\| = 0.$$

In other words, if the numerical solution approaches the analytic solution for increasingly smaller step sizes h.

For Euler's method we can establish convergence (and therefore the above heuristics are justified).

Theorem 1.10 *Euler's method is convergent.*

Proof To simplify the proof we assume that $\boldsymbol{f}$ (and therefore also $\boldsymbol{y}$) is analytic. We introduce the notation

$$\boldsymbol{e}_{n,h} = \boldsymbol{y}_{n,h} - \boldsymbol{y}(t_n),$$

the error at step n. We need to show

$$\lim_{h \to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{e}_{n,h}\| = 0.$$

Taylor's theorem for the analytic solution $\boldsymbol{y}$ gives us

$$\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \mathcal{O}(h^2).$$

Replacing $\boldsymbol{y}'$ by the ODE (24) we have

$$\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) + h\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \mathcal{O}(h^2).$$

From Euler's method we have for the numerical solution

$$\boldsymbol{y}_{n+1,h} = \boldsymbol{y}_{n,h} + h\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h}).$$

The difference of these last two expressions yields

$$\begin{aligned}\boldsymbol{e}_{n+1,h} &= \boldsymbol{y}_{n+1,h} - \boldsymbol{y}(t_{n+1}) \\ &= [\boldsymbol{y}_{n,h} + h\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h})] - \left[\boldsymbol{y}(t_n) + h\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \mathcal{O}(h^2)\right] \\ &= \boldsymbol{e}_{n,h} + h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\right] + \mathcal{O}(h^2).\end{aligned}$$

Since $\boldsymbol{y}_{n,h} = \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}$ we have

$$\boldsymbol{e}_{n+1,h} = \boldsymbol{e}_{n,h} + h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\right] + \mathcal{O}(h^2).$$

Next we can apply norms and use the triangle inequality to obtain

$$\|\boldsymbol{e}_{n+1,h}\| \leq \|\boldsymbol{e}_{n,h}\| + h\|\boldsymbol{f}(t_n, \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}) - \boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\| + ch^2.$$

Here we also used the definition of $\mathcal{O}$-notation, i.e., $g(h) = \mathcal{O}(h^p)$ if $|g(h)| \leq ch^p$ for some constant c independent of h.

Now, note that $\boldsymbol{f}$ is Lipschitz, i.e., $\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \leq \lambda\|\boldsymbol{x} - \boldsymbol{y}\|$. Taking $\boldsymbol{x} = \boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h}$ and $\boldsymbol{y} = \boldsymbol{y}(t_n)$ we obtain

$$\begin{aligned}\|\boldsymbol{e}_{n+1,h}\| &\leq \|\boldsymbol{e}_{n,h}\| + h\lambda\|\boldsymbol{y}(t_n) + \boldsymbol{e}_{n,h} - \boldsymbol{y}(t_n))\| + ch^2 \\ &= (1 + h\lambda)\|\boldsymbol{e}_{n,h}\| + ch^2.\end{aligned}$$

We can use induction to show that

$$\|\boldsymbol{e}_{n,h}\| \leq \frac{c}{\lambda}h\left[(1 + h\lambda)^n - 1\right], \qquad n = 0, 1, \ldots. \tag{7}$$

Finally, one can show that

$$(1 + h\lambda)^n < e^{nh\lambda} \leq e^{t^*\lambda} \tag{8}$$

so that

$$\|\boldsymbol{e}_{n,h}\| \leq \frac{c}{\lambda}h\left[e^{t^*\lambda} - 1\right], \qquad n = 0, 1, \ldots, N,$$

and

$$\lim_{h\to 0^+} \max_{n=0,1,\ldots,N} \|\boldsymbol{e}_{n,h}\| = \lim_{h\to 0^+} \underbrace{\frac{c}{\lambda}\left[e^{t^*\lambda} - 1\right]}_{\text{const}} h = 0.$$

∎

Remark The error estimate from the proof seems precise. In particular, since one can easily see (using the Peano kernel theorem) that $c = \max_{t\in[t_0,t0+t^*]} \|\boldsymbol{y}''\|$ works. However, it grossly over-estimates the error in many cases. Thus, it is *useless for practical purposes.*

Remark The order $\mathcal{O}(h)$ convergence of Euler's method is demonstrated in the Matlab script `EulerDemo.m`.

Example Consider the simple linear decay problem $y'(t) = -100y(t)$, $y(0) = 1$ with exact solution $y(t) = e^{-100t}$.

Since $f(t, y) = -100y$, it is clear that f is Lipschitz continuous with $\lambda = 100$ (since $\frac{\partial f}{\partial y} = -100$).

On the other hand, $y''(t) = -100y'(t) = 100^2 y(t)$, so that $c = \max \|y''\| = 100^2 = \lambda^2$.

The error estimate from the proof is of the form

$$|e_{n,h}| \leq \frac{c}{\lambda} h \left[e^{t^* \lambda} - 1 \right] = 100h \left[e^{100t^*} - 1 \right].$$

If we limit ourselves to the interval $[0, 1]$ then $t^* = 1$ and

$$|e_{n,h}| \leq 100h \left[e^{100} - 1 \right] \approx 2.6881 \times 10^{45} h.$$

On the other hand, Euler's method yields

$$\begin{aligned} y_1 &= y_0 - h100y_0 = (1 - 100h)y_0 \\ y_2 &= y_1 - h100y_1 = (1 - 100h)y_1 = (1 - 100h)^2 y_0 \\ &\vdots \\ y_n &= (1 - 100h)^n y_0 = (1 - 100h)^n, \end{aligned}$$

so that the true error is

$$|y_n - y(\underbrace{t_n}_{nh})| = \left| (1 - 100h)^n - e^{-100nh} \right| \ll 2.6881 \times 10^{45} h.$$

1.3 Trapezoidal Rule

Recall the derivation of Euler's method via integration:

$$\begin{aligned} \boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau)) d\tau \\ &\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) d\tau \\ &= \boldsymbol{y}(t_0) + (t - t_0) \boldsymbol{f}(t_0, \boldsymbol{y}(t_0)). \end{aligned}$$

FIGURE

Note that this corresponds to the "left endpoint rule" for integration. A simple – but significant – improvement over the left endpoint rule is the trapezoidal rule (for numerical integration), where we use the average of the slopes at the endpoints of the interval.

FIGURE

This leads to an improved method to solve the IVP (24), (4)

$$\begin{aligned} \boldsymbol{y}(t) &= \boldsymbol{y}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau)) d\tau \\ &\approx \boldsymbol{y}(t_0) + \int_{t_0}^{t} \frac{\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) + \boldsymbol{f}(t, \boldsymbol{y}(t))}{2} d\tau \\ &= \boldsymbol{y}(t_0) + \frac{1}{2}(t - t_0) \left[\boldsymbol{f}(t_0, \boldsymbol{y}(t_0)) + \boldsymbol{f}(t, \boldsymbol{y}(t)) \right]. \end{aligned}$$

This calculation motivates the *trapezoidal rule* (for IVPs):

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2} h \left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) \right]. \tag{9}$$

Remark The major difference between the trapezoidal rule and the Taylor/Euler methods studied earlier lies in the appearance of the "new" value of the approximate solution, $\boldsymbol{y}_{n+1}$, on *both* sides of the formula (9). This means that $\boldsymbol{y}_{n+1}$ is given only *implicitly* by equation (9), and therefore the trapezoidal rule is referred to as an *implicit* method. We will discuss one possible implementation of the trapezoidal rule later. Methods for which $\boldsymbol{y}_{n+1}$ appears only on the left-hand side of the formula are known as *explicit* methods.

The local truncation error for the trapezoidal rule can be derived by substituting the exact solution into the approximation formula (9). This leads to

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}(t_n) + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right]. \tag{10}$$

To determine the approximation error in this formula we first rearrange (10) and use the ODE (24) to replace the terms involving $\boldsymbol{f}$ by first derivatives $\boldsymbol{y}'$, i.e.,

$$\begin{aligned}&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right]\\ &\quad = \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left[\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})\right].\end{aligned}$$

Next, we replace the terms involving t_{n+1} by Taylor expansions about t_n. This leads to

$$\begin{aligned}&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left\{\boldsymbol{y}'(t_n) + \boldsymbol{y}'(t_{n+1})\right\}\\ &= \left[\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{h^2}{2}\boldsymbol{y}''(t_n) + \mathcal{O}(h^3)\right] - \boldsymbol{y}(t_n) - \tfrac{1}{2}h\left\{\boldsymbol{y}'(t_n) + \left[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \mathcal{O}(h^2)\right]\right\}\\ &= \mathcal{O}(h^3),\end{aligned}$$

so that the local truncation error of the trapezoidal rule is of order $\mathcal{O}(h^3)$. To see that the trapezoidal rule is globally a second-order method we need to establish its convergence.

Theorem 1.11 *The trapezoidal rule (9) is convergent.*

Proof Similar to the proof of convergence for Euler's method. See textbook. ∎

As mentioned earlier, the trapezoidal rule is an implicit method, and therefore, additional computational effort is required to determine the approximate solution $\boldsymbol{y}_{n+1}$ at time t_{n+1}. There are various approaches to doing this.

1. One possibility is to use a *predictor-corrector* approach. Here an explicit method (such as Euler's method) is used to *predict* a preliminary value $\tilde{\boldsymbol{y}}_{n+1}$ for $\boldsymbol{y}_{n+1}$, and the trapezoidal rule is then used in the (explicit) form

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \tilde{\boldsymbol{y}}_{n+1})\right].$$

   We will study this general approach more carefully in the context of *multistep methods* later.

2. Another approach is to use *fixed-point iteration* to compute $\boldsymbol{y}_{n+1}$. Since the trapezoidal rule is given by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{1}{2}h\left[\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right],$$

and $\boldsymbol{f}$ can be a rather general (in particular nonlinear) function, the problem of finding $\boldsymbol{y}_{n+1}$ can be rephrased as a problem of finding a root of the (system of) nonlinear equation(s)

$$G(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{z} = \boldsymbol{0}.$$

In our case

$$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \frac{h}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{z}).$$

Many techniques exist for solving such nonlinear equations such as *Newton* or *Newton-Raphson* iteration. The simplest approach is to use *functional iteration*

$$\boldsymbol{z}^{[k+1]} = \boldsymbol{g}(\boldsymbol{z}^{[k]}), \qquad k = 0, 1, 2, \ldots$$

with a good initial value $\boldsymbol{z}^{[0]}$. The famous *Banach fixed-point theorem* guarantees convergence of this approach provided the norm of the Jacobian of $\boldsymbol{g}$ is small enough, i.e.,

$$\|\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}\| < 1.$$

As a slightly weaker requirement, Lipschitz continuity of $\boldsymbol{g}$ is sufficient. Since here

$$\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} = \frac{h}{2}\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}$$

we see that – depending on the function $\boldsymbol{f}$ – the stepsize h has to be chosen small enough.

Remark The specific predictor-corrector scheme suggested in 1. above is in fact a popular numerical IVP solver in its own right. It is known under many different names such as the *classical second-order Runge-Kutta method*, the *improved Euler method*, or *Heun's method*.

Remark An implementation of the fixed-point iteration approach for the trapezoidal rule is given in the Matlab function `Trapezoid.m`. As initial guess $\boldsymbol{z}^{[0]} = \boldsymbol{y}_{n+1}^{[0]}$ we use the most recent approximate solution from the previous time step $\boldsymbol{y}_n$. Pseudocode for the algorithm is

Algorithm

Input t_0, y_0, f, h, N

$t = t_0$, $y = y_0$, $w = y$

for $n = 1$ to N do

$f_1 = f(t, w)$
$t = t + h$
for $k = 1, 2, \ldots$ do
$f_2 = f(t, w)$
$w = y + \frac{h}{2}(f_1 + f_2)$
end
$y = w$

end

Remark The order $\mathcal{O}(h^2)$ convergence of the trapezoidal rule is demonstrated in the Matlab script `TrapezoidDemo.m`.

Example The problem

$$y'(t) = \ln(3)\left(y(t) - \lfloor y(t) \rfloor - \frac{3}{2}\right), \qquad y(0) = 0,$$

can be shown to have solution

$$y(t) = -n + \frac{1}{2}\left(1 - 3^{t-n}\right), \qquad n \le t \le n+1, \quad n = 0, 1, \ldots.$$

However, since the function f here is not Lipschitz continuous we cannot expect our numerical solvers to perform as usual. The Matlab scripts `EulerFailDemo.m` and `TrapezoidFailDemo.m` show that we get about $\mathcal{O}(h^{0.8})$ convergence for *both* methods.

1.4 Theta Methods

Both Euler's method and the trapezoidal rule are included as special cases of the following formula:

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left[\theta \boldsymbol{f}(t_n \boldsymbol{y}_n) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right], \quad n = 0, 1, \ldots. \tag{11}$$

Euler's method corresponds to the choice $\theta = 1$, and the trapezoidal rule to $\theta = 1/2$. In general, formula (11) for $\theta \in [0, 1]$ is known as *theta method*.

Remark The only explicit theta method is Euler's method ($\theta = 1$), all others are implicit. Moreover, the only second-order method is the trapezoid rule. All others are first-order.

To verify the order claim we determine the local truncation error for the general theta method. As before, we insert the exact solution $\boldsymbol{y}$ into the approximate formula (11). This yields

$$\boldsymbol{y}(t_{n+1}) \approx \boldsymbol{y}(t_n) + h\left[\theta \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right]. \tag{12}$$

To determine the approximation error in this formula we proceed analogously to what we did for the trapezoidal rule. First we rearrange (12) and use the ODE (24) to replace the terms involving $\boldsymbol{f}$ by first derivatives $\boldsymbol{y}'$, i.e.,

$$\begin{aligned}
&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left[\theta \boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + (1-\theta)\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right] \\
&\quad = \boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left[\theta \boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})\right].
\end{aligned}$$

Next, we replace the terms involving t_{n+1} by Taylor expansions about t_n. This leads to

$$\begin{aligned}
&\boldsymbol{y}(t_{n+1}) - \boldsymbol{y}(t_n) - h\left\{\theta \boldsymbol{y}'(t_n) + (1-\theta)\boldsymbol{y}'(t_{n+1})\right\} \\
&= \left[\boldsymbol{y}(t_n) + h\boldsymbol{y}'(t_n) + \tfrac{h^2}{2}\boldsymbol{y}''(t_n) + \tfrac{h^3}{6}\boldsymbol{y}'''(t_n) + \mathcal{O}(h^4)\right] - \boldsymbol{y}(t_n) \\
&-h\left\{\theta \boldsymbol{y}'(t_n) + (1-\theta)\left[\boldsymbol{y}'(t_n) + h\boldsymbol{y}''(t_n) + \tfrac{h^2}{2}\boldsymbol{y}'''(t_n) + \mathcal{O}(h^3)\right]\right\} \\
&= \left(\theta - \tfrac{1}{2}\right) h^2 \boldsymbol{y}''(t_n) + \left(\tfrac{1}{2}\theta - \tfrac{1}{3}\right) h^3 \boldsymbol{y}'''(t_n) + \mathcal{O}(h^4),
\end{aligned}$$

so that the local truncation error of the general theta method is of order $\mathcal{O}(h^2)$. However, for $\theta = 1/2$ the first term on the right drops out, and we have a local truncation error of order $\mathcal{O}(h^3)$.

Convergence of the general theta method is established in Exercise 1.1 (see Assignment 2).

Remark The choice $\theta = 0$ yields the so-called *backward Euler method* which has particularly nice stability properties, and is often used to solve *stiff* equations. Other choices of θ are used less frequently.

2 Multistep Methods

Up to now, all methods we studied were single step methods, i.e., the value $\boldsymbol{y}_{n+1}$ was found using information only from the previous time level t_n. Now we will consider so-called *multistep methods*, i.e., more of the history of the solution will affect the value $\boldsymbol{y}_{n+1}$.

2.1 Adams Methods

Consider the first-order ODE

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)).$$

If we integrate from t_{n+1} to t_{n+2} we have

$$\int_{t_{n+1}}^{t_{n+2}} \boldsymbol{y}'(\tau)d\tau = \int_{t_{n+1}}^{t_{n+2}} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau$$

or

$$\boldsymbol{y}(t_{n+2}) - \boldsymbol{y}(t_{n+1}) = \int_{t_{n+1}}^{t_{n+2}} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau. \tag{13}$$

As we saw earlier for Euler's method and for the trapezoidal rule, different numerical integration rules lead to different ODE solvers. In particular, the left-endpoint rule yields Euler's method, while the trapezoidal rule for integration gives rise to the trapezoidal rule for IVPs. Incidentally, the right-endpoint rule provides us with the backward Euler method.

We now use a different quadrature formula for the integral in (13).

Example Instead of viewing the slope $\boldsymbol{f}$ as a constant on the interval $[t_n, t_{n+1}]$ we now represent $\boldsymbol{f}$ by its *linear interpolating polynomial* at the points $\tau = t_n$ and $\tau = t_{n+1}$ given in Lagrange form, i.e.,

$$\begin{aligned} \boldsymbol{p}(\tau) &= \frac{\tau - t_{n+1}}{t_n - t_{n+1}}\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \frac{\tau - t_n}{t_{n+1} - t_n}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1})) \\ &= \frac{t_{n+1} - \tau}{h}\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \frac{\tau - t_n}{h}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1})), \end{aligned}$$

where we have used the stepsize $t_{n+1} - t_n = h$.

FIGURE

Therefore, the integral becomes

$$\begin{aligned} \int_{t_{n+1}}^{t_{n+2}} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau &\approx \int_{t_{n+1}}^{t_{n+2}} \boldsymbol{p}(\tau)d\tau \\ &= \int_{t_{n+1}}^{t_{n+2}} \left[\frac{t_{n+1} - \tau}{h}\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)) + \frac{\tau - t_n}{h}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\right] d\tau \\ &= \left[\boldsymbol{f}(t_n, \boldsymbol{y}(t_n))\left(-\frac{1}{2}\right)\frac{(t_{n+1} - \tau)^2}{h} + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1}))\frac{(\tau - t_n)^2}{2h}\right]_{t_{n+1}}^{t_{n+2}} \\ &= \frac{3h}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1})) - \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{y}(t_n)). \end{aligned}$$

Thus (13) motivates the numerical method

$$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[3\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \boldsymbol{f}(t_n, \boldsymbol{y}_n)\right]. \tag{14}$$

Since formula (14) involves two previously computed solution values, this method is known as a *two-step method*. More precisely, is is known as the *second-order Adams-Bashforth method* (or AB method) dating back to 1883.

Remark 1. We will establish later that this method is indeed of second order accuracy.

2. Note that the method (14) requires two initial conditions. Since the IVP will give us only one initial condition, in the Matlab demo script `ABDemo.m` we take the second starting value from the exact solution. This is, of course, not realistic, and in practice one often precedes the Adams-Bashforth method by one step of, e.g., a second-order Runge-Kutta method (see later). However, even a single Euler step (which is also of order $\mathcal{O}(h^2)$) can also be used to start up (and maintain the accuracy of) the second-order AB method. This approach can also be used in `ABDemo.m` by uncommenting the corresponding line.

Example The Matlab script `ABDemo.m` compares the convergence of Euler's method (the one-step AB method) with the two-step AB method (14) for the IVP

$$y'(t) = -y^2(t), \qquad y(0) = 1$$

on the interval $[0, 10]$ with different stepsizes $N = 50, 100, 200$ and 400. The exact solution of this problem is

$$y(t) = \frac{1}{t+1}.$$

Example If we use a linear Lagrange interpolant to the integrand $\boldsymbol{f}$ of (13) at the points $\tau = t_{n+1}$ and $\tau = t_{n+2}$ then we obtain

$$\boldsymbol{y}(t_{n+2}) \approx y(t_{n+1}) + \frac{h}{2}\left[\boldsymbol{f}(t_{n+1}, \boldsymbol{y}(t_{n+1})) + \boldsymbol{f}(t_{n+2}, \boldsymbol{y}(t_{n+2}))\right]$$

or the numerical scheme

$$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2})\right]. \tag{15}$$

FIGURE

This method is known as second-order *Adams-Moulton method* (or AM method). It is a one-step method, and identical to the trapezoidal rule studied earlier (modulo a shift of the indices by one).

Remark In general, a pth-order Adams method is obtained by replacing the integrand $\boldsymbol{f}$ in (13) by a polynomial of degree $p-1$. However, the Adams-Bashforth method is an explicit method that uses the most recent information as well as $p-1$ "historical" points to fit the polynomial to. The pth-order Adams-Moulton method is an implicit method that fits the polynomial to the point to be determined next, the current point, and $p-2$ "historical" points. Therefore, the pth-order AB method is a p-step method, while the pth-order AM method is a $p-1$-step method.

For a general s-step Adams method we start the derivation as usual with the first-order ODE (24) and integrate from the current time t_{n+s-1} to the new time t_{n+s}. This gives us

$$\boldsymbol{y}(t_{n+s}) - \boldsymbol{y}(t_{n+s-1}) = \int_{t_{n+s-1}}^{t_{n+s}} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau))d\tau. \tag{16}$$

As mentioned above, for the s-step AB method we now fit the slope $\boldsymbol{f}$ with a polynomial of degree $s-1$ at the s "known" points $t_n, t_{n+1}, \ldots, t_{n+s-2}, t_{n+s-1}$, i.e., we replace the integrand in (16) by the polynomial

$$\boldsymbol{p}(\tau) = \sum_{m=0}^{s-1} p_m(\tau)\boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{m+n}),$$

where the p_m are the Lagrange functions (cf. Section 1.0.2)

$$p_m(\tau) = \prod_{\substack{\ell=0 \\ \ell\neq m}}^{s-1} \frac{\tau - t_{n+\ell}}{t_{n+m} - t_{n+\ell}}, \qquad m = 0, 1, \ldots, s-1.$$

This gives us

$$\begin{aligned}
\boldsymbol{y}(t_{n+s}) - \boldsymbol{y}(t_{n+s-1}) &\approx \int_{t_{n+s-1}}^{t_{n+s}} \boldsymbol{p}(\tau)d\tau \\
&= \int_{t_{n+s-1}}^{t_{n+s}} \sum_{m=0}^{s-1} p_m(\tau)\boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{m+n})d\tau \\
&= \sum_{m=0}^{s-1} \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{m+n}) \int_{t_{n+s-1}}^{t_{n+s}} p_m(\tau)d\tau.
\end{aligned}$$

The calculations just performed motivate the numerical method

$$\boldsymbol{y}_{n+s} = \boldsymbol{y}_{n+s-1} + h\sum_{m=0}^{s-1} b_m \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{m+n}), \tag{17}$$

where the coefficients b_m, $m = 0, 1, \ldots, s-1$, are given by (using the substitution $u = \tau - t_{n+s-1}$, and the fact that $t_{n+s} - t_{n+s-1} = h$)

$$\begin{aligned}
b_m &= \frac{1}{h}\int_{t_{n+s-1}}^{t_{n+s}} p_m(\tau)d\tau \\
&= \frac{1}{h}\int_0^h p_m(t_{n+s-1} + u)du.
\end{aligned} \tag{18}$$

Formula (17) together with the coefficients (18) is known as the general *s-step Adams-Bashforth method*.

Example The coefficients b_m are independent of the specific stepsize h and timestep n, so they can be computed once and for all. For the case $s = 2$ discussed earlier we compute

$$p_0(\tau) = \frac{\tau - t_{n+1}}{t_n - t_{n+1}},$$

Order	Formula	LTE
1	$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}_n$	$\frac{h^2}{2}\boldsymbol{y}''(\eta)$
2	$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[3\boldsymbol{f}_{n+1} - \boldsymbol{f}_n\right]$	$\frac{5h^3}{12}\boldsymbol{y}'''(\eta)$
3	$\boldsymbol{y}_{n+3} = \boldsymbol{y}_{n+2} + \frac{h}{12}\left[23\boldsymbol{f}_{n+2} - 16\boldsymbol{f}_{n+1} + 5\boldsymbol{f}_n\right]$	$\frac{3h^4}{8}\boldsymbol{y}^{(4)}(\eta)$
4	$\boldsymbol{y}_{n+4} = \boldsymbol{y}_{n+3} + \frac{h}{24}\left[55\boldsymbol{f}_{n+3} - 59\boldsymbol{f}_{n+2} + 37\boldsymbol{f}_{n+1} - 9\boldsymbol{f}_n\right]$	$\frac{251h^5}{720}\boldsymbol{y}^{(5)}(\eta)$
5	$\boldsymbol{y}_{n+5} = \boldsymbol{y}_{n+4} + \frac{h}{720}\left[1901\boldsymbol{f}_{n+4} - 2774\boldsymbol{f}_{n+3} + 2616\boldsymbol{f}_{n+2} - 1274\boldsymbol{f}_{n+1} + 251\boldsymbol{f}_n\right]$	$\frac{95h^6}{2888}\boldsymbol{y}^{(6)}(\eta)$

Table 1: Adams-Bashforth formulas of different order. Notation: $\boldsymbol{f}_{n+m}$ denotes $\boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m})$, $m = 0, 1, \ldots, 5$, LTE stands for local truncation error.

and

$$\begin{aligned} b_0 &= \frac{1}{h}\int_0^h \frac{t_{n+1} + u - t_{n+1}}{t_n - t_{n+1}} du \\ &= \frac{1}{h}\int_0^h \frac{u}{-h} du \\ &= -\frac{1}{h^2}\left[\frac{u^2}{2}\right]_0^h = -\frac{1}{2}. \end{aligned}$$

Formulas for other choices of s are listed in Table 1.

Remark 1. The technique of using an interpolating polynomial $\boldsymbol{p}$ of degree $s-1$ at s equally spaced nodes with spacing h to replace the integrand $\boldsymbol{f}$ leads to so-called *Newton-Cotes formulas* for numerical integration. It is known that the interpolation error in this case is of the order $\mathcal{O}(h^s)$, and integration of this polynomial over an interval of length h adds another factor of h to the order. Therefore, the s-step Adams-Bashforth method has a local truncation error of order $\mathcal{O}(h^{s+1})$, which – provided the method converges – translates into a global method of order s.

2. General Adams-Moulton formulas can be derived similarly and are listed in Table 2. Note that the backward Euler method does not quite fit the general description of an AM method, since it is a single step method of order 1 (while the other AM methods are s-step methods of order $s+1$). In fact, there are two single step AM methods: the backward Euler method and the trapezoidal rule.

2.2 The Predictor-Corrector Idea

As mentioned earlier, one way to implement an implicit scheme is to couple it with a corresponding explicit scheme of the same order. We will now explain this *predictor-corrector* approach using the 2nd-order AB and AM methods.

Order	Formula	LTE
1	$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}_{n+1}$	$-\frac{h^2}{2}\boldsymbol{y}''(\eta)$
2	$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}_{n+2} + \boldsymbol{f}_{n+1}\right]$	$-\frac{h^3}{12}\boldsymbol{y}'''(\eta)$
3	$\boldsymbol{y}_{n+3} = \boldsymbol{y}_{n+2} + \frac{h}{12}\left[5\boldsymbol{f}_{n+3} + 8\boldsymbol{f}_{n+2} - \boldsymbol{f}_{n+1}\right]$	$-\frac{h^4}{24}\boldsymbol{y}^{(4)}(\eta)$
4	$\boldsymbol{y}_{n+4} = \boldsymbol{y}_{n+3} + \frac{h}{24}\left[9\boldsymbol{f}_{n+4} + 19\boldsymbol{f}_{n+3} - 5\boldsymbol{f}_{n+2} + \boldsymbol{f}_{n+1}\right]$	$-\frac{19h^5}{720}\boldsymbol{y}^{(5)}(\eta)$
5	$\boldsymbol{y}_{n+5} = \boldsymbol{y}_{n+4} + \frac{h}{720}\left[251\boldsymbol{f}_{n+5} + 646\boldsymbol{f}_{n+4} - 264\boldsymbol{f}_{n+3} + 106\boldsymbol{f}_{n+2} - 19\boldsymbol{f}_{n+1}\right]$	$-\frac{3h^6}{160}\boldsymbol{y}^{(6)}(\eta)$

Table 2: Adams-Moulton formulas of different order. Notation: $\boldsymbol{f}_{n+m}$ denotes $\boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m})$, $m = 0, 1, \ldots, 5$, LTE stands for local truncation error.

We start with the *predictor* – in our case the second-order AB method. However, we treat its output only as a temporary answer, i.e.,

$$\tilde{\boldsymbol{y}}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[3\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \boldsymbol{f}(t_n, \boldsymbol{y}_n)\right].$$

Next we *correct* this value by using it on the right-hand side of the second-order AM method, i.e.,

$$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \boldsymbol{f}(t_{n+2}, \tilde{\boldsymbol{y}}_{n+2})\right].$$

While this approach provides a simple realization of an implicit method, it can also be used to create a scheme that uses a variable stepsize h. The basic idea is to use the difference $|\tilde{\boldsymbol{y}}_{n+2} - \boldsymbol{y}_{n+2}|$ to judge the accuracy of the method. The following algorithm describes the general idea:

Algorithm

$\tilde{\boldsymbol{y}}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[3\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \boldsymbol{f}(t_n, \boldsymbol{y}_n)\right]$

$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \boldsymbol{f}(t_{n+2}, \tilde{\boldsymbol{y}}_{n+2})\right]$

$\kappa = \frac{1}{6}|\tilde{\boldsymbol{y}}_{n+2} - \boldsymbol{y}_{n+2}|$

if κ is relatively large, then

    $h \leftarrow h/2$ (i.e., reduce the stepsize)

    repeat

else if κ is relatively small, then

    $h \leftarrow 2h$ (i.e., increase the stepsize)

else

    continue (i.e., keep h)

end

The specific choice of e in the algorithm is motivated by the following argument. The local truncation errors for the second-order AB and AM methods, respectively, are

$$\begin{aligned} \boldsymbol{y}(t_{n+2}) - \tilde{\boldsymbol{y}}_{n+2} &= \frac{5}{12}h^3\boldsymbol{y}'''(\eta_{AB}) \\ \boldsymbol{y}(t_{n+2}) - \boldsymbol{y}_{n+2} &= -\frac{1}{12}h^3\boldsymbol{y}'''(\eta_{AM}). \end{aligned}$$

If we assume that $\boldsymbol{y}'''$ is nearly constant over the interval of interest, i.e., $\boldsymbol{y}'''(\eta_{AB}) \approx \boldsymbol{y}'''(\eta_{AM}) \approx \boldsymbol{y}'''(\eta)$, then we can subtract the above two equations from each other to get

$$\boldsymbol{y}_{n+2} - \tilde{\boldsymbol{y}}_{n+2} \approx \frac{1}{2}h^3\boldsymbol{y}'''(\eta),$$

and therefore the error at this time step is

$$|\boldsymbol{y}(t_{n+2}) - \boldsymbol{y}_{n+2}| \approx \frac{1}{12}h^3\boldsymbol{y}'''(\eta) \approx \frac{1}{6}|\boldsymbol{y}_{n+2} - \tilde{\boldsymbol{y}}_{n+2}|.$$

Remark 1. Finding a good way to characterize "relatively large" and "relatively small" in the algorithm can be tricky.

2. Note that it may be necessary to generate additional function values by interpolation if the stepsize is reduced and the predictor has to be evaluated for this new stepsize.

3. A variable stepsize, variable-order AB-AM predictor-corrector scheme is implemented in Matlab in the routine `ode113`.

2.3 Order and Convergence of Multistep Methods

There are even more general multistep methods than the Adams methods. We will write them in the form

$$\sum_{m=0}^{s} a_m \boldsymbol{y}_{n+m} = h\sum_{m=0}^{s} b_m \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m}), \qquad n = 0, 1, \ldots, \tag{19}$$

where the coefficients a_m and b_m, $m = 0, 1, \ldots, s$ are independent of h, n, and the underlying ODE. Usually, the formula is normalized so that $a_s = 1$. The formula is a true s-step formula if either a_0 or b_0 are different from zero. Different choices of the coefficients a_m and b_m yield different numerical methods. In particular, if $b_s = 0$ the method will be explicit. Otherwise it will be implicit.

Remark The general multistep formula (19) is of the same form as so-called *recursive* or *infinite impulse response (IIR) digital filters* used in digital signal processing.

Example The second-order AB method corresponds to $s = 2$ with

$$a_2 = 1,\ a_1 = -1,\ a_0 = 0,\ b_2 = 0,\ b_1 = 3/2,\ b_0 = -1/2,$$

and the second-order AM method corresponds to $s = 2$ with

$$a_2 = 1,\ a_1 = -1,\ a_0 = 0,\ b_2 = 1/2,\ b_1 = 1/2,\ b_0 = 0.$$

Remark The coefficients a_m and b_m will play a crucial role in our following discussion of order and convergence of multistep methods, as well as later on in our study of stability.

As used many times before, a numerical IVP solver of the form

$$\boldsymbol{y}_{n+1} = \mathcal{Y}_n(\boldsymbol{f}, h, \boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$$

is of order p if and only if

$$\boldsymbol{y}(t_{n+1}) - \mathcal{Y}_n(\boldsymbol{f}, h, \boldsymbol{y}(t_0), \boldsymbol{y}(t_1), \ldots, \boldsymbol{y}(t_n)) = \mathcal{O}(h^{p+1}).$$

For the multistep methods (19) we can alternatively consider the linear functional ψ_t defined by

$$\begin{aligned} \psi_t \boldsymbol{y} &= \sum_{m=0}^{s} a_m \boldsymbol{y}(t+mh) - h \sum_{m=0}^{s} b_m \boldsymbol{f}(t+mh, \boldsymbol{y}(t+mh)) \\ &= \sum_{m=0}^{s} a_m \boldsymbol{y}(t+mh) - h \sum_{m=0}^{s} b_m \boldsymbol{y}'(t+mh). \end{aligned}$$

Then the s-step method (19) is of order p if and only if

$$\psi_t \boldsymbol{y} = \mathcal{O}(h^{p+1})$$

for all sufficiently smooth functions $\boldsymbol{y}$.

We now characterize the order p of a multistep method in terms of the coefficients a_m and b_m.

Theorem 2.1 *The multistep method*

$$\sum_{m=0}^{s} a_m \boldsymbol{y}_{n+m} = h \sum_{m=0}^{s} b_m \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m})$$

is of order $p \geq 1$ if and only if

$$\begin{aligned} \sum_{m=0}^{s} a_m &= 0, \\ \sum_{m=0}^{s} \frac{m^k}{k!} a_m &= \sum_{m=0}^{s} \frac{m^{k-1}}{(k-1)!} b_m, \qquad k = 1, 2, \ldots, p, \\ \sum_{m=0}^{s} \frac{m^{p+1}}{(p+1)!} a_m &\neq \sum_{m=0}^{s} \frac{m^p}{p!} b_m. \end{aligned}$$

Proof We have

$$\psi_t \boldsymbol{y} = \sum_{m=0}^{s} a_m \boldsymbol{y}(t+mh) - h \sum_{m=0}^{s} b_m \boldsymbol{y}'(t+mh).$$

Using Taylor expansions for both $\boldsymbol{y}$ and $\boldsymbol{y}'$ we obtain

$$\begin{aligned}
\psi_t\boldsymbol{y} &= \sum_{m=0}^{s} a_m \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{y}^{(k)}(t)(mh)^k - h\sum_{m=0}^{s} b_m \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{y}^{(k+1)}(t)(mh)^k \\
&= \sum_{m=0}^{s} a_m \sum_{k=0}^{\infty} \frac{1}{k!}\boldsymbol{y}^{(k)}(t)m^k h^k - h\sum_{m=0}^{s} b_m \sum_{k=1}^{\infty} \frac{1}{(k-1)!}\boldsymbol{y}^{(k)}(t)m^{k-1}h^{k-1} \\
&= \sum_{k=0}^{\infty}\left(\sum_{m=0}^{s}\frac{m^k}{k!}a_m\right)\boldsymbol{y}^{(k)}(t)h^k - \sum_{k=1}^{\infty}\left(\sum_{m=0}^{s}\frac{m^{k-1}}{(k-1)!}b_m\right)\boldsymbol{y}^{(k)}(t)h^k \\
&= \left(\sum_{m=0}^{s} a_m\right)\boldsymbol{y}(t) + \sum_{k=1}^{\infty}\left(\sum_{m=0}^{s}\frac{m^k}{k!}a_m\right)\boldsymbol{y}^{(k)}(t)h^k - \sum_{k=1}^{\infty}\left(\sum_{m=0}^{s}\frac{m^{k-1}}{(k-1)!}b_m\right)\boldsymbol{y}^{(k)}(t)h^k \\
&= \left(\sum_{m=0}^{s} a_m\right)\boldsymbol{y}(t) + \sum_{k=1}^{\infty}\left(\sum_{m=0}^{s}\frac{m^k}{k!}a_m - \sum_{m=0}^{s}\frac{m^{k-1}}{(k-1)!}b_m\right)\boldsymbol{y}^{(k)}(t)h^k
\end{aligned}$$

We get $\psi_t\boldsymbol{y} = \mathcal{O}(h^{p+1})$ by satisfying the conditions as claimed. ∎

Remark 1. We can now use the simple conditions in Theorem 2.1 to check the order of *any* multistep method. This is generally *much easier* than the special techniques we used earlier (see the example below).

2. A method is called *consistent* if it is of order $p \geq 1$, i.e., if

$$\sum_{m=0}^{s} a_m = 0 \quad \text{and} \quad \sum_{m=0}^{s} m a_m = \sum_{m=0}^{s} b_m.$$

We noted earlier that merely establishing the order of a method does not ensure its convergence (see the second example below). With this new terminology we can say that *consistency alone does not imply convergence.*

3. If we introduce the polynomials (often called *characteristic polynomials* or *generating polynomials* of the method)

$$\rho(w) = \sum_{m=0}^{s} a_m w^m, \qquad \sigma(w) = \sum_{m=0}^{s} b_m w^m,$$

then one can show that the general multistep method is of order p if and only if there exists a constant $c \neq 0$ such that

$$\rho(w) - \sigma(w)\ln w = c(w-1)^{p+1} + \mathcal{O}(|w-1|^{p+2}) \qquad \text{as } w \to 1. \tag{20}$$

Note that this condition tells us how well the log-function is approximated by the rational function $\frac{\rho}{\sigma}$ near $w = 1$. In terms of the polynomials ρ and σ the two consistency conditions above correspond to

$$\rho(1) = 0 \quad \text{and} \quad \rho'(1) = \sigma(1).$$

4. The constant c in (20) (as well as the difference of the two sides of the third condition in Theorem 2.1) is in fact the local error constant of the method (cf. the following example and the local truncation errors listed in Tables 1 and 2).

Example We show that the Adams-Bashforth method (14) derived earlier is indeed of second order. The iteration formula was

$$\boldsymbol{y}_{n+2} - \boldsymbol{y}_{n+1} = \frac{h}{2}\left[3\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \boldsymbol{f}(t_n, \boldsymbol{y}_n)\right],$$

so that – as noted earlier – $s = 2$ and $a_2 = 1$, $a_1 = -1$, $a_0 = 0$, $b_2 = 0$, $b_1 = 3/2$, $b_0 = -1/2$.

Now,

$$\sum_{m=0}^{2} a_m = 0 - 1 + 1 = 0,$$

and for $k = 1$ (note: $0! = 1! = 1$)

$$\begin{aligned} \sum_{m=0}^{2} m a_m &= \sum_{m=0}^{2} b_m \\ \iff 0(0) + (1)(-1) + 2(1) &= -\frac{1}{2} + \frac{3}{2} + 0 \\ \iff 1 &= 1, \end{aligned}$$

for $k = 2$

$$\begin{aligned} \sum_{m=0}^{2} \frac{m^2}{2} a_m &= \sum_{m=0}^{2} m b_m \\ \iff \frac{0}{2}(0) + \frac{1}{2}(-1) + \frac{4}{2}(1) &= (0)(-\frac{1}{2}) + (1)\frac{3}{2} + (2)0 \\ \iff \frac{3}{2} &= \frac{3}{2}, \end{aligned}$$

and for $k = 3$

$$\begin{aligned} \sum_{m=0}^{2} \frac{m^3}{3!} a_m &= \sum_{m=0}^{2} \frac{m^2}{2!} b_m \\ \iff \frac{0}{6}(0) + \frac{1}{6}(-1) + \frac{8}{6}(1) &= \frac{0}{2}(-\frac{1}{2}) + \frac{1}{2}\frac{3}{2} + \frac{4}{2}0 \\ \iff \frac{7}{6} &= \frac{3}{4}. \end{aligned}$$

Therefore, the method is indeed of order $p = 2$. Moreover, the difference $\frac{7}{6} - \frac{3}{4} = \frac{5}{12}$ (the error constant listed in Table 1).

Alternatively, we can check the condition

$$\rho(w) - \sigma(w)\ln w = c(w-1)^{p+1} + \mathcal{O}(|w-1|^{p+2}) \qquad \text{as } w \to 1.$$

For our example

$$\begin{aligned}\rho(w) &= \sum_{m=0}^{2} a_m w^m = w^2 - w,\\ \sigma(w) &= \sum_{m=0}^{2} b_m w^m = \frac{3}{2}w - \frac{1}{2}.\end{aligned}$$

Since the right-hand side of our condition is written in terms of $w-1$ we express everything in terms of $\xi = w-1$ and then take the limit as $\xi \to 0$. Thus

$$\begin{aligned}\rho(\xi) &= (\xi+1)^2 - (\xi+1) = \xi^2 + \xi,\\ \sigma(\xi) &= \frac{3}{2}(\xi+1) - \frac{1}{2} = \frac{3}{2}\xi + 1,\end{aligned}$$

and therefore (using the Taylor expansion of the logarithm)

$$\begin{aligned}\rho(\xi) - \sigma(\xi)\ln(\xi+1) &= \left(\xi^2+\xi\right) - \left(\frac{3}{2}\xi+1\right)\ln(\xi+1)\\ &= \left(\xi^2+\xi\right) - \left(\frac{3}{2}\xi+1\right)\sum_{k=1}^{\infty}(-1)^{k-1}\frac{\xi^k}{k}\\ &= \left(\xi^2+\xi\right) - \left(\frac{3}{2}\xi+1\right)\left(\xi - \frac{\xi^2}{2} + \frac{\xi^3}{3} - \cdots\right)\\ &= \left(\xi^2+\xi\right) - \left(\xi + \xi^2 - \frac{5}{12}\xi^3 + \mathcal{O}(\xi^4)\right) = \frac{5}{12}\xi^3 + \mathcal{O}(\xi^4).\end{aligned}$$

Thus, $c = \frac{5}{12} \neq 0$ (again, the error constant of Table 1), and the order is $p = 2$ as before.

Example The implicit 2-step method

$$\boldsymbol{y}_{n+2} - 3\boldsymbol{y}_{n+1} + 2\boldsymbol{y}_n = h\left[\frac{13}{12}\boldsymbol{f}(t_{n+2},\boldsymbol{y}_{n+2}) - \frac{5}{3}\boldsymbol{f}(t_{n+1},\boldsymbol{y}_{n+1}) - \frac{5}{12}\boldsymbol{f}(t_n,\boldsymbol{y}_n)\right] \quad (21)$$

has order two. This can easily be verified using the criteria of Theorem 2.1. However, consider the trivial IVP

$$y'(t) = 0, \qquad y(0) = 1,$$

with solution $y(t) = 1$. For this example the right-hand side of (21) is always zero, so we immediately get the 3-term recurrence relation

$$y_{n+2} - 3y_{n+1} + 2y_n = 0.$$

The general solution of this equation is given by

$$y_n = c_1 + c_2 2^n, \qquad n = 0, 1, \ldots, \quad (22)$$

with arbitrary constants c_1 and c_2. This can easily be verified using induction.

While the choice $c_1 = 1$ and $c_2 = 0$ does provide the exact solution, in practice we will have to initialize the method with two values y_0 and y_1, and most likely these

values will be different (which is equivalent to having $c_2 \neq 0$ in (22) above). In this case, however, the numerical solution blows up (since $2^n \to \infty$ for $n \to \infty$). Thus, only for one very special set of starting values do we have an accurate solution. In general, the method *does not converge* (even though it is consistent, and is even of order 2).

We are now ready to present a necessary and sufficient condition for convergence of a multistep method that is very easy to check. To this end we have

Definition 2.2 *A (complex) polynomial p obeys the* root condition *if*

- *all its zeros, i.e., all z such that $p(z) = 0$, lie in the unit disk, i.e., $|z| \leq 1$, and*
- *all zeros on the unit circle are simple, i.e., if $|z| = 1$ then $p'(z) \neq 0$.*

Theorem 2.3 *(Dahlquist Equivalence Theorem) Consider the general multistep method (cf. (19))*

$$\sum_{m=0}^{s} a_m \boldsymbol{y}_{n+m} = h \sum_{m=0}^{s} b_m \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m}), \qquad n = 0, 1, \ldots,$$

and assume that the starting values $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{s-1}$ are accurate, i.e., they are determined up to an error that tends to zero for $h \to 0$. Then the multistep method is convergent if and only if it is consistent and the polynomial ρ obeys the root condition.

Proof The proof is too involved to be included here. ∎

Remark

1. An immediate – and very important – consequence of the Equivalence Theorem is that a multistep method whose characteristic polynomial ρ does not satisfy the root condition cannot be convergent. Do not use such methods!
2. We will see later that the fact that ρ satisfies the root condition is equivalent to stability of a multistep method. Therefore, for multistep methods,

$$\text{convergence} \quad \Longleftrightarrow \quad \text{consistency \& stability.}$$

Example Earlier we claimed that the implicit 2-step method

$$\boldsymbol{y}_{n+2} - 3\boldsymbol{y}_{n+1} + 2\boldsymbol{y}_n = h\left[\frac{13}{12}\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}) - \frac{5}{3}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \frac{5}{12}\boldsymbol{f}(t_n, \boldsymbol{y}_n)\right] \tag{23}$$

has order two, and gave a counterexample to show that it cannot be convergent. Now we can use the Dahlquist Equivalence Theorem to establish this fact. The characteristic polynomial ρ of the method is given by

$$\rho(w) = \sum_{m=0}^{2} a_m w^m = w^2 - 3w + 2 = (w-1)(w-2),$$

and we see that ρ violates the root condition since one of its zeros, $w = 2$, lies outside the unit disk. According to the Equivalence Theorem it cannot be convergent.

Example The characteristic polynomials ρ for all Adams methods (both AB and AM) are of the same type, namely,

$$\rho(w) = w^s - w = w(w^{s-1} - 1),$$

and so they all satisfy the root condition. As a consequence, *all Adams methods are convergent* (since we already established that they are consistent).

If we look at the general s-step method

$$\sum_{m=0}^{s} a_m \boldsymbol{y}_{n+m} = h \sum_{m=0}^{s} b_m \boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m}), \qquad n = 0, 1, \ldots,$$

then we see that this method involves $2s + 1$ free parameters (after normalization). Therefore, one might think that it is possible to construct an s-step method that is of order $2s$. Unfortunately, there is another theorem by Dahlquist that states that one cannot have a convergent s-step method of order $2s$ for any $s \geq 3$. More precisely,

Theorem 2.4 *(Dahlquist's First Barrier) The maximal order of a convergent s-step method is at most*

- $s + 2$ *for implicit schemes with s even,*
- $s + 1$ *for implicit schemes with s odd, and*
- s *for explicit schemes.*

Proof Also too complicated. ∎

Remark 1. A procedure for construction of a convergent s-step method of order $s + 1$ is outlined in the textbook.

2. Adams-Bashforth methods are *optimal* in the sense that the corresponding order is as high as possible. The same is true for Adams-Moulton formulas with odd s. It can be shown that implicit s-step methods of order $s + 2$ are of questionable stability.

2.4 Backward Differentiation Formulae

As mentioned earlier, the "extreme" choice $\rho(w) = w^{s-1}(w - 1)$ that always satisfies the root condition and also places as many of the zeros at the origin yields the family of Adams methods. If we, on the other hand, choose an extreme $\sigma(w) = \beta w^s$, then we obtain the so-called *backward differentiation formulae* (BDFs).

These methods have their name from the way in which they can be derived. Starting from the ODE $\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y})$ we do not integrate as before (and fit the integrand $\boldsymbol{f}$ on the right-hand side by a polynomial), but instead fit the derivative $\boldsymbol{y}'$ on the left-hand side by the derivative of the interpolating polynomial to the $s + 1$ data points chosen *backward* from the new point, i.e., $(t_{n+s}, \boldsymbol{y}_{n+s}), (t_{n+s-1}, \boldsymbol{y}_{n+s-1}), \ldots, (t_n, \boldsymbol{y}_n)$.

Example In the simplest case, $s = 1$, we get the *Backward Euler method*. The interpolating polynomial to the data $(t_{n+1}, \boldsymbol{y}_{n+1}), (t_n, \boldsymbol{y}_n)$ is given by

$$\boldsymbol{p}(\tau) = \frac{t_{n+1} - \tau}{t_{n+1} - t_n}\boldsymbol{y}_n + \frac{\tau - t_{n+1}}{t_{n+1} - t_n}\boldsymbol{y}_{n+1},$$

and its derivative is

$$\boldsymbol{p}'(\tau) = -\frac{1}{h}\boldsymbol{y}_n + \frac{1}{h}\boldsymbol{y}_{n+1},$$

where we have use $t_{n+1} - t_n = h$ as usual. If we replace the right-hand side of the ODE by $\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})$, then we end up with the numerical method

$$-\frac{1}{h}\boldsymbol{y}_n + \frac{1}{h}\boldsymbol{y}_{n+1} = \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) \quad \Longleftrightarrow \quad \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})$$

– the backward Euler method.

In general, one can show that once the form of the polynomial σ and the order s are chosen, then the method is determined.

Lemma 2.5 *For a BDF of order s with $\sigma(w) = \beta w^s$ we have*

$$\beta = \left(\sum_{m=1}^{s} \frac{1}{m}\right)^{-1}, \quad \textit{and} \quad \rho(w) = \beta \sum_{m=1}^{s} \frac{1}{m} w^{s-m}(w-1)^m.$$

Proof Straightforward algebra using (20) and a Taylor series expansion for the logarithm. See textbook. ∎

Example In the simplest case $s = 1$ we have $\sigma(w) = \beta w$ with $\beta = 1$ and $\rho(w) = w - 1$. This gives us the backward Euler method mentioned above.

For $s = 2$ we have $\sigma(w) = \beta w^2$, and

$$\beta = \left(\sum_{m=1}^{2} \frac{1}{m}\right)^{-1} = \frac{1}{1 + \frac{1}{2}} = \frac{2}{3}.$$

With this value

$$\rho(w) = \beta \sum_{m=1}^{2} \frac{1}{m} w^{s-m}(w-1)^m = \frac{2}{3}\left[w(w-1) + \frac{1}{2}(w-1)^2\right] = w^2 - \frac{4}{3}w + \frac{1}{3}.$$

This results in the BDF formula

$$\boldsymbol{y}_{n+2} - \frac{4}{3}\boldsymbol{y}_{n+1} + \frac{1}{3}\boldsymbol{y}_n = \frac{2}{3}h\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}).$$

While the Adams methods were constructed so that they satisfy the root condition, this is no longer automatically true for the BDFs. In fact, we have

Theorem 2.6 *The characteristic polynomial ρ for a BDF satisfies the root condition and the underlying BDF method is convergent if and only if $1 \leq s \leq 6$.*

Order	Formula	LTE
1	$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}_{n+1}$	$-\frac{h^2}{2}\boldsymbol{y}''(\eta)$
2	$\boldsymbol{y}_{n+2} - \frac{4}{3}\boldsymbol{y}_{n+1} + \frac{1}{3}\boldsymbol{y}_n = \frac{2h}{3}\boldsymbol{f}_{n+2}$	$-\frac{2h^3}{9}\boldsymbol{y}'''(\eta)$
3	$\boldsymbol{y}_{n+3} - \frac{18}{11}\boldsymbol{y}_{n+2} + \frac{9}{11}\boldsymbol{y}_{n+1} - \frac{2}{11}\boldsymbol{y}_n = \frac{6h}{11}\boldsymbol{f}_{n+3}$	$-\frac{3h^4}{22}\boldsymbol{y}^{(4)}(\eta)$
4	$\boldsymbol{y}_{n+4} - \frac{48}{25}\boldsymbol{y}_{n+3} + \frac{36}{25}\boldsymbol{y}_{n+2} - \frac{16}{25}\boldsymbol{y}_{n+1} + \frac{3}{25}\boldsymbol{y}_n = \frac{12h}{25}\boldsymbol{f}_{n+4}$	$-\frac{12h^5}{125}\boldsymbol{y}^{(5)}(\eta)$
5	$\boldsymbol{y}_{n+5} - \frac{300}{137}\boldsymbol{y}_{n+4} + \frac{300}{137}\boldsymbol{y}_{n+3} - \frac{200}{137}\boldsymbol{y}_{n+2} + \frac{75}{137}\boldsymbol{y}_{n+1} - \frac{12}{137}\boldsymbol{y}_n = \frac{60h}{137}\boldsymbol{f}_{n+5}$	$-\frac{10h^6}{137}\boldsymbol{y}^{(6)}(\eta)$
6	$\boldsymbol{y}_{n+6} - \frac{360}{147}\boldsymbol{y}_{n+5} + \frac{450}{147}\boldsymbol{y}_{n+4} - \frac{400}{147}\boldsymbol{y}_{n+3} + \frac{225}{147}\boldsymbol{y}_{n+2} - \frac{72}{147}\boldsymbol{y}_{n+1} + \frac{10}{147}\boldsymbol{y}_n = \frac{60h}{147}\boldsymbol{f}_{n+6}$	$-\frac{20h^7}{343}\boldsymbol{y}^{(7)}(\eta)$

Table 3: Backward differentiation formulas of different order. Notation: $\boldsymbol{f}_{n+m}$ denotes $\boldsymbol{f}(t_{n+m}, \boldsymbol{y}_{n+m})$, $m = 0, 1, \ldots, 6$, LTE stands for local truncation error.

Proof Too involved.

Remark 1. The root condition fails for $s > 6$.

2. The coefficients and local truncation errors for all 6 BDFs of practical interest are listed in Table 3.

3. Note that all BDF methods are *implicit* methods, and therefore special care is needed for their implementation. However, as the following example shows (and as we will see in more detail later), they have better stability properties than, say AB methods, and are therefore better suited for *stiff* problems.

Example The Matlab script `BDFDemo.m` compares the performance of the second-order BDF method with that of the second-order AB method for the linear – but stiff – ODE system

$$\boldsymbol{y}'(t) = \begin{bmatrix} -10 & 1 \\ 0 & -1 \end{bmatrix} \boldsymbol{y}, \qquad \boldsymbol{y}(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The solution to this problem is

$$\boldsymbol{y}(t) = \begin{bmatrix} \frac{1}{9}e^{-t} + \frac{8}{9}e^{-10t} \\ e^{-t} \end{bmatrix},$$

and the presence of the two different scales in the first component of the solution is what makes this problem *stiff*. We will discuss stiff problems in more detail in Section 4.

3 Runge-Kutta Methods

In contrast to the multistep methods of the previous section Runge-Kutta methods are single-step methods - however, with multiple *stages* per step. They are motivated by the dependence of the Taylor methods on the specific IVP. These new methods do not require derivatives of the right-hand side function $\boldsymbol{f}$ 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. Modern developments are mostly due to John Butcher in the 1960s.

3.1 Second-Order Runge-Kutta Methods

As always we consider the general first-order ODE system

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)). \tag{24}$$

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

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

The first derivative can be replaced by the right-hand side of the differential equation (24), and the second derivative is obtained by differentiating (24), i.e.,

$$\begin{aligned} \boldsymbol{y}''(t) &= \boldsymbol{f}_t(t,\boldsymbol{y}) + \boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{y}'(t) \\ &= \boldsymbol{f}_t(t,\boldsymbol{y}) + \boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{f}(t,\boldsymbol{y}), \end{aligned}$$

with Jacobian $\boldsymbol{f}_{\boldsymbol{y}}$. We will from now on neglect the dependence of $\boldsymbol{y}$ on t when it appears as an argument to $\boldsymbol{f}$. Therefore, the Taylor expansion becomes

$$\begin{aligned} \boldsymbol{y}(t+h) &= \boldsymbol{y}(t) + h\boldsymbol{f}(t,\boldsymbol{y}) + \frac{h^2}{2}\left[\boldsymbol{f}_t(t,\boldsymbol{y}) + \boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{f}(t,\boldsymbol{y})\right] + \mathcal{O}(h^3) \\ &= \boldsymbol{y}(t) + \frac{h}{2}\boldsymbol{f}(t,\boldsymbol{y}) + \frac{h}{2}\left[\boldsymbol{f}(t,\boldsymbol{y}) + h\boldsymbol{f}_t(t,\boldsymbol{y}) + h\boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{f}(t,\boldsymbol{y})\right] + \mathcal{O}(h^3) \end{aligned} \tag{25}$$

Recalling the multivariate Taylor expansion

$$\boldsymbol{f}(t+h, \boldsymbol{y}+\boldsymbol{k}) = \boldsymbol{f}(t,\boldsymbol{y}) + h\boldsymbol{f}_t(t,\boldsymbol{y}) + \boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{k} + \ldots$$

we see that the expression in brackets in (25) can be interpreted as

$$\boldsymbol{f}(t+h, \boldsymbol{y}+h\boldsymbol{f}(t,\boldsymbol{y})) = \boldsymbol{f}(t,\boldsymbol{y}) + h\boldsymbol{f}_t(t,\boldsymbol{y}) + h\boldsymbol{f}_{\boldsymbol{y}}(t,\boldsymbol{y})\boldsymbol{f}(t,\boldsymbol{y}) + \mathcal{O}(h^2).$$

Therefore, we get

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

or the numerical method

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left(\frac{1}{2}\boldsymbol{k}_1 + \frac{1}{2}\boldsymbol{k}_2\right), \tag{26}$$

with

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n), \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_1). \end{aligned}$$

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

Remark 1. The $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ are known as *stages* of the Runge-Kutta method. They correspond to different estimates for the *slope* of the solution. Note that $\boldsymbol{y}_n + h\boldsymbol{k}_1$ corresponds to an Euler step with stepsize h starting from $(t_n, \boldsymbol{y}_n)$. Therefore, $\boldsymbol{k}_2$ corresponds to the slope of the solution one would get by taking one Euler step with stepsize h starting from $(t_n, \boldsymbol{y}_n)$. The numerical method (26) now consists of a single step with the average of the slopes $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$.

2. The notation used here differs slightly from that used in the textbook. There the stages are defined differently. I find the interpretation in terms of slopes more intuitive.

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

$$\boldsymbol{y}(t+h) = \boldsymbol{y}(t) + h\left[b_1\tilde{\boldsymbol{k}}_1 + b_2\tilde{\boldsymbol{k}}_2\right] + \mathcal{O}(h^3) \tag{27}$$

with

$$\begin{aligned} \tilde{\boldsymbol{k}}_1 &= \boldsymbol{f}(t, \boldsymbol{y}) \\ \tilde{\boldsymbol{k}}_2 &= \boldsymbol{f}(t + c_2 h, \boldsymbol{y} + h a_{21}\tilde{\boldsymbol{k}}_1). \end{aligned}$$

Clearly, this is a generalization of the classical Runge-Kutta method since the choice $b_1 = b_2 = \frac{1}{2}$ and $c_2 = a_{21} = 1$ yields that case.

It is customary to arrange the coefficients a_{ij}, b_i, and c_i in a so-called *Runge-Kutta* or *Butcher tableaux* as follows:

$$\begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{b}^T. \end{array}$$

Accordingly, the Butcher tableaux for the classical second-order Runge-Kutta method is

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2}. \end{array}$$

Explicit Runge-Kutta methods are characterized by a strictly lower triangular matrix A, i.e., $a_{ij} = 0$ if $j \geq i$. Moreover, the coefficients c_i and a_{ij} are connected by the condition

$$c_i = \sum_{j=1}^{\nu} a_{ij}, \qquad i = 1, 2, \ldots, \nu.$$

This says that c_i is the row sum of the i-th row of the matrix A. This condition is required to have a method of order one. We limit our discussion to such methods now.

Thus, for an explicit second-order method we necessarily have $a_{11} = a_{12} = a_{22} = c_1 = 0$. We can now study what other combinations of b_1, b_2, c_2 and a_{21} in (27) give us a second-order method. The bivariate Taylor expansion yields

$$\begin{aligned} \boldsymbol{f}(t + c_2 h, \boldsymbol{y} + h a_{21} \tilde{\boldsymbol{k}}_1) &= \boldsymbol{f}(t, \boldsymbol{y}) + c_2 h \boldsymbol{f}_t(t, \boldsymbol{y}) + h a_{21} \boldsymbol{f}_{\boldsymbol{y}}(t, \boldsymbol{y}) \tilde{\boldsymbol{k}}_1 + \mathcal{O}(h^2) \\ &= \boldsymbol{f}(t, \boldsymbol{y}) + c_2 h \boldsymbol{f}_t(t, \boldsymbol{y}) + h a_{21} \boldsymbol{f}_{\boldsymbol{y}}(t, \boldsymbol{y}) \boldsymbol{f}(t, \boldsymbol{y}) + \mathcal{O}(h^2). \end{aligned}$$

Therefore, the general second-order Runge-Kutta assumption (27) becomes

$$\begin{aligned} \boldsymbol{y}(t + h) &= \boldsymbol{y}(t) + h \left[b_1 \boldsymbol{f}(t, \boldsymbol{y}) + b_2 \left\{ \boldsymbol{f}(t, \boldsymbol{y}) + c_2 h \boldsymbol{f}_t(t, \boldsymbol{y}) + h a_{21} \boldsymbol{f}_{\boldsymbol{y}}(t, \boldsymbol{y}) \boldsymbol{f}(t, \boldsymbol{y}) \right\} \right] + \mathcal{O}(h^3) \\ &= \boldsymbol{y}(t) + (b_1 + b_2) h \boldsymbol{f}(t, \boldsymbol{y}) + b_2 h^2 \left[c_2 \boldsymbol{f}_t(t, \boldsymbol{y}) + a_{21} \boldsymbol{f}_{\boldsymbol{y}}(t, \boldsymbol{y}) \boldsymbol{f}(t, \boldsymbol{y}) \right] + \mathcal{O}(h^3). \end{aligned}$$

In order for this to match the general Taylor expansion (25) we want

$$\begin{aligned} b_1 + b_2 &= 1 \\ c_2 b_2 &= \tfrac{1}{2} \\ a_{21} b_2 &= \tfrac{1}{2}. \end{aligned}$$

Thus, we have a system of three nonlinear equations for our four unknowns. One popular solution is the choice $b_1 = 0$, $b_2 = 1$, and $c_2 = a_{21} = \frac{1}{2}$. This leads to the *modified Euler method* or the (explicit) *midpoint rule*

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \boldsymbol{k}_2$$

with

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2} \boldsymbol{k}_1). \end{aligned}$$

Its Butcher tableaux is of the form

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ \hline & 0 & 1. \end{array}$$

Remark The choice $b_1 = 1$, $b_2 = 0$ leads to Euler's method. However, since now $c_2 b_2 \neq \frac{1}{2}$ and $a_{21} b_2 \neq \frac{1}{2}$ this method does not have second-order accuracy.

General explicit Runge-Kutta methods are of the form

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \sum_{j=1}^{\nu} b_j \boldsymbol{k}_j$$

with

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\
\boldsymbol{k}_2 &= \boldsymbol{f}(t_n + c_2 h, \boldsymbol{y}_n + a_{21} h \boldsymbol{k}_1) \\
&\vdots \\
\boldsymbol{k}_\nu &= \boldsymbol{f}(t_n + c_\nu h, \boldsymbol{y}_n + h \sum_{j=1}^{\nu-1} a_{\nu,j} \boldsymbol{k}_j).
\end{aligned}$$

Determination of the coefficients is rather complicated. We now describe (without derivation) the most famous Runge-Kutta method.

3.2 Fourth-Order Runge-Kutta Methods

The classical method is given by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h \left[\frac{\boldsymbol{k}_1}{6} + \frac{\boldsymbol{k}_2}{3} + \frac{\boldsymbol{k}_3}{3} + \frac{\boldsymbol{k}_4}{6} \right] \tag{28}$$

with

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\
\boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_1\right) \\
\boldsymbol{k}_3 &= \boldsymbol{f}\left(t_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_2\right) \\
\boldsymbol{k}_4 &= \boldsymbol{f}\left(t_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_3\right).
\end{aligned}$$

Its Butcher tableaux is of the form

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}.
\end{array}$$

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 $\boldsymbol{f}$ per time step.

Remark We saw earlier that in each time step of the second-order Runge-Kutta method we need to perform two evaluations of $\boldsymbol{f}$, and for a fourth-order method there are four evaluations. More generally, one can observe the situation described in Table 4.

These data imply that higher-order (> 4) Runge-Kutta methods are relatively inefficient. Precise data for higher-order methods does not seem to be known. 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 (see Section 5).

evaluations of $\boldsymbol{f}$ per time step	2	3	4	5	6	7	8	9	10	11
maximum order achievable	2	3	4	4	5	6	6	7	7	8

Table 4: Efficiency of Runge-Kutta methods.

3.3 Connection to Numerical Integration Rules

We now illustrate the connection of Runge-Kutta methods to numerical integration rules.

As before, we consider the IVP

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t, \boldsymbol{y}(t)) \\ \boldsymbol{y}(t_0) &= \boldsymbol{y}_0 \end{aligned}$$

and integrate both sides of the differential equation from t to $t+h$ to obtain

$$\boldsymbol{y}(t+h) - \boldsymbol{y}(t) = \int_t^{t+h} \boldsymbol{f}(\tau, \boldsymbol{y}(\tau)) d\tau. \tag{29}$$

Therefore, the solution to our IVP can be obtained by solving the integral equation (29). Of course, we can use numerical integration to do this:

1. Using the left endpoint method

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

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

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

   since $\tau = t$, the left endpoint of the interval. Thus, as we saw earlier, (29) is equivalent to Euler's method.

2. Using the trapezoidal rule

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

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

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

   The corresponding IVP solver is therefore

   $$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2} \boldsymbol{f}(t_n, \boldsymbol{y}_n) + \frac{h}{2} \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}).$$

Note that this is *not* equal to the classical second-order Runge-Kutta method since we have a $\boldsymbol{y}_{n+1}$ 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 replace $\boldsymbol{y}_{n+1}$ on the right-hand side by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_n, \boldsymbol{y}_n).$$

Then we end up with the method

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \frac{h}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_n + h\boldsymbol{f}(t_n, \boldsymbol{y}_n))$$

or

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left[\frac{1}{2}\boldsymbol{k}_1 + \frac{1}{2}\boldsymbol{k}_2\right]$$

with

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_1), \end{aligned}$$

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 the fourth-order Runge-Kutta method in case there is no dependence of $\boldsymbol{f}$ on $\boldsymbol{y}$.

5. Gauss quadrature leads to so-called *Gauss-Runge-Kutta* or *Gauss-Legendre* methods. One such method is the implicit midpoint rule

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_n + \frac{h}{2}, \frac{1}{2}(\boldsymbol{y}_n + \boldsymbol{y}_{n+1}))$$

encountered earlier. The Butcher tableaux for this one-stage order two method is given by

$$\begin{array}{c|c} \frac{1}{2} & \frac{1}{2} \\ \hline & 1. \end{array}$$

Note that the general implicit Runge-Kutta method is of the form

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\sum_{j=1}^{\nu} b_j \boldsymbol{k}_j$$

with

$$\boldsymbol{k}_j = \boldsymbol{f}(t_n + c_j h, \boldsymbol{y}_n + h\sum_{i=1}^{j} a_{j,i}\boldsymbol{k}_i)$$

for all values of $j = 1, \ldots, \nu$. Thus, the implicit midpoint rule corresponds to

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{k}_1$$

with

$$\boldsymbol{k}_1 = \boldsymbol{f}(t_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_1)$$

– obviously an implicit method.

6. More general *implicit Runge-Kutta methods* exist. However, their construction is more difficult, and can sometimes be linked to *collocation methods*. Some details are given at the end of Chapter 3 in the textbook.

4 Stiffness and Stability

There are two basic notions of stability. The first notion of stability is concerned with the behavior of the numerical solution for a fixed value $t > 0$ as $h \to 0$.

Definition 4.1 *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.*

For a consistent s-step method one can show that the notion of stability and the fact that its characteristic polynomial ρ satisfies the root condition are equivalent. Therefore, as mentioned earlier, for an s-step method we have

$$\text{convergence} \quad \Longleftrightarrow \quad \text{consistent \& stable.}$$

Remark Sometimes this notion of stability is referred to as *zero stability.*

This concept of stability also plays an important role in determining the global truncation error. In fact, for a convergent (consistent and stable) method the local truncation errors add up as expected, i.e., a convergent s-step method with $\mathcal{O}(h^{p+1})$ local truncation error has a global error of order $\mathcal{O}(h^p)$.

4.1 Linear Stability Analysis

A second notion of stability is concerned with the behavior of the solution as $t \to \infty$ with a fixed stepsize h. This notion of stability is often referred to as *absolute stability*, and it is important when dealing with *stiff* ODEs.
A Model Problem: For $\lambda \in \mathbb{R}$ we consider the family of scalar *linear* initial value problems (the discussion in the textbook uses the analogous system case)

$$\begin{aligned} y'(t) &= \lambda y(t), \quad t \in [0, T], \\ y(0) &= y_0. \end{aligned}$$

The solution of this problem is given by

$$y(t) = y_0 e^{\lambda t}.$$

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

$$y_\delta(0) = y_0 + \delta.$$

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

$$y_\delta(t) = (y_0 + \delta)e^{\lambda t}.$$

Therefore, if $\lambda \leq 0$, 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.*

An *absolutely 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 behaves for the stable model problem above, i.e., in the case $\lambda \leq 0$. Euler's method states that

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) \\ &= y_n + h\lambda y_n \\ &= (1 + \lambda h) y_n. \end{aligned}$$

Therefore, by induction,

$$y_n = (1 + \lambda h)^n y_0.$$

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 real $\lambda < 0$ 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.

The set of λh for which the growth factor is less than one is called the *linear stability domain* $\mathcal{D}$.

Example For Euler's method we have

$$|1 + \lambda h| < 1$$

so that (for complex λ)

$$\mathcal{D}_{Euler} = \{z = \lambda h \in \mathbb{C} : \ |z + 1| < 1\},$$

a rather small circular subset of the left half of the complex plane.

FIGURE

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

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$$

is *unconditionally stable* for the above problem.

To see this we have

$$y_{n+1} = y_n + h\lambda y_{n+1}$$

or

$$(1 - \lambda h) y_{n+1} = y_n.$$

Therefore,

$$y_n = \left(\frac{1}{1 - \lambda h}\right)^n y_0.$$

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. Of course, this statement pertains only to the stability of the method. In order to achieve an appropriate accuracy, h still has to be chosen reasonably small. However, as we will see below, we do not have to worry about stepsize constraints imposed by the *stiffness* of the problem.

The linear stability domain for the backward Euler method is given by the entire negative complex half-plane, i.e.,

$$\mathcal{D}_{backwardEuler} = \{z = \lambda h \in \mathbb{C} : \ \mathrm{Re} z < 0\} = \mathbb{C}^-.$$

In general, absolute stability of a linear multistep formula can be determined with the help of its characteristic polynomials. In fact, an s-step method is absolutely stable if the roots of the polynomial $\phi = \rho - \lambda h \sigma$ lie inside the unit disk. Here ρ and σ are defined as earlier, i.e.,

$$\begin{aligned} \rho(w) &= \sum_{m=0}^{s} a_m w^m \\ \sigma(w) &= \sum_{m=0}^{s} b_m w^m \end{aligned}$$

and a_m and b_m are the coefficients of the s-step method

$$\sum_{m=0}^{s} a_m y_{n+m} = h \sum_{m=0}^{s} b_m f(t_{n+m}, y_{n+m})$$

with $f(t_{n+m}, y_{n+m}) = \lambda y_{n+m}$, $m = 0, \ldots, s$, given by the model problem.

The linear stability domain (also known as the *region of absolute stability*) of an s-step method is then the region in the complex plane for which the roots of the polynomial $\phi = \rho - \lambda h \sigma$ lie inside the unit disk.

Example For Euler's method $\rho(w) = w - 1$, and $\sigma(w) = 1$, so that

$$\phi(w) = \rho(w) - \lambda h \sigma(w) = (w - 1) - \lambda h = w - (1 + \lambda h)$$

with root $1 + \lambda h$. The region of absolute stability $\mathcal{D}$ is therefore

$$\mathcal{D}_{Euler} = \{z = \lambda h \in \mathbb{C} : \ |1 + z| < 1\},$$

the same region we found earlier.

Remark The ideas discussed above for scalar linear IVPs can be extended to both nonlinear equations and systems of equations. Many of the properties of a method for the scalar linear case carry over to the more complicated cases.

4.2 Stiffness

The phenomenon of *stiffness* is not precisely defined in the literature. Some attempts at describing a stiff problem are:

- A problem is stiff if it contains widely varying time scales, i.e., some components of the solution decay much more rapidly than others.
- A problem is stiff if the stepsize is dictated by stability requirements rather than by accuracy requirements.
- A problem is stiff if explicit methods don't work, or work only extremely slowly.
- A linear problem is stiff if all of its eigenvalues have negative real part, and the *stiffness ratio* (the ratio of the magnitudes of the real parts of the largest and smallest eigenvalues) is large.
- More generally, a problem is stiff if the eigenvalues of the Jacobian of $\boldsymbol{f}$ differ greatly in magnitude.

Example A stiff ODE is illustrated in the Matlab script `StiffDemo2.m`. If one lights a match, the ball of flame grows rapidly until it reaches critical size. Then it remains at that size because the amount of oxygen being consumed by the combustion in the interior of the ball balances the amount available through the surface. The model is

$$y'(t) = y(t)^2 - y(t)^3, \quad y(0) = \delta, \quad t \in [0, 2/\delta].$$

The scalar quantity $y(t)$ represents the radius of the ball of flame at time t. The y^2 and y^3 terms in the model come from the surface area and volume, respectively. The critical parameter is the initial radius, δ, which is "small". The solution is sought on an interval of time inversely proportional to δ. The solution to this problem will grow very slowly until t reaches $1/\delta$, then a quick transition occurs to a value close to 1, where the solution then remains. The exact solution

$$y(t) = \frac{1}{W(ae^{a-t}) + 1},$$

where W is the *Lambert W* function (the solution of the equation $W(z)e^{W(z)} = z$) and $a = 1/\delta - 1$ (see the Maple worksheet `StiffDemo2.mws`).

Note how it takes the "non-stiff" solver `ode45` in `StiffDemo2.m` longer and longer to obtain a solution for decreasing values of δ.

This problem is initially not stiff, but becomes so as the solution approaches the steady state of 1.

Remark

1. Stiff ODEs arise in various applications; e.g., when modeling chemical reactions, in control theory, or electrical circuits, such as the *Van der Pol equation* in relaxation oscillation (see the Matlab script `VanderPolDemo.m`).
2. One way to deal with a stiff problem is to have an *unconditionally stable* solver.

4.3 *A*-Stability

Unconditional stability, in fact, unconditional stability for the model problem

$$y'(t) \quad = \quad \lambda y(t), \quad t \in [0, T]$$

$$y(0) \quad = \quad y_0$$

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

Definition 4.2 *An s-step method is A-*stable *if its region of absolute stability includes the entire negative real axis.*

Remark This is covered in the textbook in the form of Lemma 4.7.

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

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}).$$

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

Theorem 4.3 *If a linear s-step method is A-stable then it must be an implicit method. Moreover, the order of the method is at most 2.*

Theorem 4.3 implies that the only other s-step Adams method we need to consider is the *implicit trapezoid method* (or second-order Adams-Moulton method) introduced earlier:

$$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}) + \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right]. \tag{30}$$

To see that the implicit trapezoid method is A-stable we consider

$$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\lambda\left[\boldsymbol{y}_{n+2} + \boldsymbol{y}_{n+1}\right]$$

or

$$(1 - \lambda\frac{h}{2})\boldsymbol{y}_{n+2} = (1 + \lambda\frac{h}{2})\boldsymbol{y}_{n+1}.$$

Therefore, as can be verified easily by induction,

$$\boldsymbol{y}_n = \left(\frac{1 + \lambda\frac{h}{2}}{1 - \lambda\frac{h}{2}}\right)^n \boldsymbol{y}_0,$$

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 $h > 0$. In other words, the linear stability domain for the trapezoidal rule is

$$\mathcal{D}_{TR} = \{z = \lambda h \in \mathbb{C} : \ \mathrm{Re}z < 0\} = \mathbb{C}^-.$$

Another method admitted by the Second Dahlquist Barrier is the second-order BDF method. It can also be shown to be A-stable.

Furthermore, it can be shown that no explicit Runge-Kutta methods can be A-stable. However, all implict Gauss-Legendre (Runge-Kutta) methods – such as the implicit midpoint rule – are A-stable.

Remark 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 fall into one of the following two categories:

1. Higher-order BDF methods are almost A-stable (see Figure 4.4 in the textbook). *Gear methods* are based on such BDF methods, and achieve higher-order along with numerical stability for stiff problems 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 Matlab, Gear methods are implemented in the stiff solver `ode15s`. In Maple some of these methods are available through the `method=lsode` option (invoking the Livermore ODEPACK by Hindmarsh) to `dsolve,numeric`.

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

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).

5 Error Control

5.1 The Milne Device and Predictor-Corrector Methods

We already discussed the basic idea of the predictor-corrector approach in Section 2. In particular, there we gave the following algorithm that made use of the 2nd-order AB and AM (trapezoid) methods.

Algorithm

$\tilde{\boldsymbol{y}}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[3\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) - \boldsymbol{f}(t_n, \boldsymbol{y}_n)\right]$

$\boldsymbol{y}_{n+2} = \boldsymbol{y}_{n+1} + \frac{h}{2}\left[\boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1}) + \boldsymbol{f}(t_{n+2}, \tilde{\boldsymbol{y}}_{n+2})\right]$

$\kappa = \frac{1}{6}|\tilde{\boldsymbol{y}}_{n+2} - \boldsymbol{y}_{n+2}|$

if κ is relatively large, then

$h \leftarrow h/2$ (i.e., reduce the stepsize)

repeat

else if κ is relatively small, then

$h \leftarrow 2h$ (i.e., increase the stepsize)

else

continue (i.e., keep h)

end

The basic idea of this algorithm is captured in a framework known as the *Milne device* (see flowchart on p.77 of the textbook). Earlier we explained how we arrived at the formula for the estimate, κ, of the local truncation error in the special case above.

For a general pair of explicit AB and implicit AM methods of (the same) order p we have local truncation errors of the form

$$\boldsymbol{y}(t_{n+s}) - \tilde{\boldsymbol{y}}_{n+s} = \tilde{c}h^{p+1}\boldsymbol{y}^{(p+1)}(\eta_{AB}) \tag{31}$$

$$\boldsymbol{y}(t_{n+s}) - \boldsymbol{y}_{n+s} = ch^{p+1}\boldsymbol{y}^{(p+1)}(\eta_{AM}). \tag{32}$$

Note that (as in the earlier case of 2nd-order methods) we have shifted the indices for the two methods so that they align, i.e., the value to be determined at the current time step has subscript $n+s$.

If we assume that the derivative $\boldsymbol{y}^{(p+1)}$ is nearly constant over the interval of interest, i.e., $\boldsymbol{y}^{(p+1)}(\eta_{AB}) \approx \boldsymbol{y}^{(p+1)}(\eta_{AM}) \approx \boldsymbol{y}^{(p+1)}(\eta)$, then we can subtract equation (32) from equation (31) to get

$$\boldsymbol{y}_{n+s} - \tilde{\boldsymbol{y}}_{n+s} \approx (\tilde{c} - c)h^{p+1}\boldsymbol{y}^{(p+1)}(\eta),$$

and therefore

$$h^{p+1}\boldsymbol{y}^{(p+1)}(\eta) \approx \frac{1}{\tilde{c} - c}\left(\boldsymbol{y}_{n+s} - \tilde{\boldsymbol{y}}_{n+s}\right).$$

If we sustitute this expression back into (32) we get an estimate for the error at this time step as

$$\|\boldsymbol{y}(t_{n+2})-\boldsymbol{y}_{n+2}\| = |c|h^{p+1}\|\boldsymbol{y}^{(p+1)}(\eta_{AM})\| \approx |c|h^{p+1}\|\boldsymbol{y}^{(p+1)}(\eta)\| \approx \left|\frac{c}{\tilde{c}-c}\right| \|\boldsymbol{y}_{n+s}-\tilde{\boldsymbol{y}}_{n+s}\|.$$

Thus, in the flowchart, the *Milne estimator* κ is of the form

$$\kappa = \left|\frac{c}{\tilde{c}-c}\right| \|\boldsymbol{y}_{n+s}-\tilde{\boldsymbol{y}}_{n+s}\|.$$

The reason for the upper bound $h\delta$ on κ for the purpose of stepsize adjustments (instead of simply a tolerance δ for the global error) is motivated by the heuristics that the transition from the local truncation error to the global error reduces the local error by an order of h.

Remark 1. We point out that the use of the (explicit) predictor serves two purposes here. First, it eliminates the use of iterative methods to cope with the implicitness of the corrector, and secondly – for the same price – we also get an estimator for the local error that allows us to use variable stepsizes, and thus compute the solution more efficiently. Sometimes the error estimate κ is even added as a correction (or extrapolation) to the numerical solution $\boldsymbol{y}_{n+s}$. However, this process rests on a somewhat shaky theoretical foundation, since one cannot guarantee that the resulting value really is more accurate.

2. As mentioned earlier, since an s-step Adams method requires startup values at equally spaced points, it may be necessary to compute these values via polynomial interpolation (see the "remeshing" steps in the flowchart).

5.2 Richardson Extrapolation

Another simple, but rather inefficient, way to estimate the local error κ (and then again use the general framework of the flowchart to obtain an adaptive algorithm) is to look at two numerical approximations coming from the *same* method: one based on a single step with stepsize h, and the other based on two steps with stepsize $h/2$. We first describe the general idea of *Richardson extrapolation*, and then illustrate the idea on the example of Euler's method.

Whenever we approximate a quantity F by a numerical approximation scheme F_h with a formula of the type

$$F = F_h + \underbrace{\mathcal{O}(h^p)}_{=E_h}, \qquad p \geq 1, \tag{33}$$

we can use an extrapolation method to combine already computed values (at stepsizes h and $h/2$) to obtain a better estimate.

Assume we have computed two approximate values F_h (using stepsize h) and $F_{h/2}$ (using 2 steps with stepsize $h/2$) for the desired quantity F. Then the error for the stepsize $h/2$ satisfies

$$E_{h/2} \approx c\left(\frac{h}{2}\right)^p = c\frac{h^p}{2^p} \approx \frac{1}{2^p}E_h.$$

Therefore, using (33),

$$F - F_{h/2} = E_{h/2} \approx \frac{1}{2^p} E_h = \frac{1}{2^p}(F - F_h).$$

This implies

$$F\left(1 - \frac{1}{2^p}\right) \approx F_{h/2} - \frac{1}{2^p} F_h$$

or

$$F \approx \frac{2^p}{2^p - 1}\left[F_{h/2} - \frac{F_h}{2^p}\right].$$

The latter can be rewritten as

$$F \approx \frac{2^p}{2^p - 1} F_{h/2} - \frac{1}{2^p - 1} F_h. \tag{34}$$

This is the *Richardson extrapolation* formula.

Since the error E_h is given by $F - F_h$, and F in turn can be approximated via (34) we also obtain an *estimate for the error* E_h, namely

$$E_h = F - F_h \approx \frac{2^p}{2^p - 1} F_{h/2} - \frac{1}{2^p - 1} F_h - F_h = \frac{2^p}{2^p - 1}\left[F_{h/2} - F_h\right].$$

We can use this in place of κ and obtain an adaptive algorithm following the flowchart.

Example We know that Euler's method

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(t_n, \boldsymbol{y}_n)$$

produces a solution that is accurate up to terms of order $\mathcal{O}(h)$ so that $p = 1$. If we denote by $\boldsymbol{y}_{n+1,h}$ the solution obtained taking one step with step size h from t_n to t_{n+1}, and by $\boldsymbol{y}_{n+1,h/2}$ the solution obtained taking two steps with step size h from t_n to t_{n+1}, then we can use

$$\boldsymbol{y}_{n+1} = 2\boldsymbol{y}_{n+1,h/2} - \boldsymbol{y}_{n+1,h}$$

to improve the accuracy of Euler's method, or to obtain the error estimate

$$\kappa = \|\boldsymbol{y}_{n+1,h/2} - \boldsymbol{y}_{n+1,h}\|.$$

5.3 Embedded Runge-Kutta Methods

For (explicit) Runge-Kutta methods another strategy exists for obtaining adaptive solvers: the so-called *embedded Runge-Kutta methods*. With an embedded Runge-Kutta method we also compute the value $\boldsymbol{y}_{n+1}$ 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.

In order to see what the local error estimate κ looks like we assume we have the (lower order) method that produces a solution $\boldsymbol{y}_{n+1}$ such that

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}(t_{n+1}) + \boldsymbol{c}h^{p+1} + \mathcal{O}(h^{p+2}),$$

where $\boldsymbol{y}$ is the exact (local) solution based on the initial condition $\boldsymbol{y}(t_n) = \boldsymbol{y}_n$. Similarly, the higher-order method produces a solution $\tilde{\boldsymbol{y}}_{n+1}$ such that

$$\tilde{\boldsymbol{y}}_{n+1} = \boldsymbol{y}(t_{n+1}) + \mathcal{O}(h^{p+2}).$$

Subtraction of the second of these equations from the first yields

$$\boldsymbol{y}_{n+1} - \tilde{\boldsymbol{y}}_{n+1} \approx \boldsymbol{c}h^{p+1},$$

which is a decent approximation of the error of the lower-order method. Therefore, we have

$$\kappa = \|\boldsymbol{y}_{n+1} - \tilde{\boldsymbol{y}}_{n+1}\|.$$

Example One of the simplest examples of an embedded Runge-Kutta method is the following second-third-order scheme defined by its (combined) Butcher tableaux

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{2}{3} & \frac{2}{3} & 0 & 0 \\ \frac{2}{3} & 0 & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & \frac{3}{4} & 0 \\ \hline & \frac{1}{4} & \frac{3}{8} & \frac{3}{8}. \end{array}$$

This implies that the second-order method is given by

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left[\frac{1}{4}\boldsymbol{k}_1 + \frac{3}{4}\boldsymbol{k}_2\right]$$

with

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(t_n + \frac{2}{3}h, \boldsymbol{y}_n + \frac{2}{3}h\boldsymbol{k}_1), \end{aligned}$$

and the third-order method looks like

$$\tilde{\boldsymbol{y}}_{n+1} = \boldsymbol{y}_n + h\left[\frac{1}{4}\boldsymbol{k}_1 + \frac{3}{8}\boldsymbol{k}_2 + \frac{3}{8}\boldsymbol{k}_3\right]$$

with the *same* $\boldsymbol{k}_1$ and $\boldsymbol{k}_2$ and

$$\boldsymbol{k}_3 = \boldsymbol{f}(t_n + \frac{2}{3}h, \boldsymbol{y}_n + \frac{2}{3}h\boldsymbol{k}_2).$$

Now, the local error estimator is given by

$$\begin{aligned} \kappa &= \|\boldsymbol{y}_{n+1} - \tilde{\boldsymbol{y}}_{n+1}\| \\ &= \frac{3}{8}h\|\boldsymbol{k}_2 - \boldsymbol{k}_3\|. \end{aligned}$$

Now we can again use the adaptive strategy outlined in the flowchart, and have an *adaptive* second-order Runge-Kutta method that uses only three function evaluations per time step.

Remark 1. Sometimes people use the higher-order solution $\tilde{\boldsymbol{y}}_{n+1}$ as their numerical approximation "justified" by the argument that this solution is obtained with a higher-order method. However, a higher-order method need not be more accurate than a lower-order method.

2. Another example of a second-third-order embedded Runge-Kutta method is implemented in Matlab as `ode23`. However, its definition is more complicated since the third-order method uses the final computed value of the second-order method as its initial slope.

Example We now describe the classical fourth-fifth-order Runge-Kutta-Fehlberg method which was first published in 1970. An implementation of this method is included in nearly all numerical software packages. In Matlab the adaptive RKF45 method can be accessed using the function `ode45`.

The fourth-order method is an "inefficient" one that uses five function evaluations at each time step. Specifically,

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\left[\frac{25}{216}\boldsymbol{k}_1 + \frac{1408}{2565}\boldsymbol{k}_3 + \frac{2197}{4104}\boldsymbol{k}_4 - \frac{1}{5}\boldsymbol{k}_5\right]$$

with

$$\begin{aligned}
\boldsymbol{k}_1 &= \boldsymbol{f}(t_n, \boldsymbol{y}_n), \\
\boldsymbol{k}_2 &= \boldsymbol{f}\left(t_n + \frac{h}{4}, \boldsymbol{y}_n + \frac{h}{4}\boldsymbol{k}_1\right), \\
\boldsymbol{k}_3 &= \boldsymbol{f}\left(t_n + \frac{3}{8}h, \boldsymbol{y}_n + \frac{3h}{32}\boldsymbol{k}_1 + \frac{9h}{32}\boldsymbol{k}_2\right), \\
\boldsymbol{k}_4 &= \boldsymbol{f}\left(t_n + \frac{12}{13}h, \boldsymbol{y}_n + \frac{1932h}{2197}\boldsymbol{k}_1 - \frac{7200h}{2197}\boldsymbol{k}_2 + \frac{7296h}{2197}\boldsymbol{k}_3\right), \\
\boldsymbol{k}_5 &= \boldsymbol{f}\left(t + h, \boldsymbol{y}_n + \frac{439h}{216}\boldsymbol{k}_1 - 8h\boldsymbol{k}_2 + \frac{3680h}{513}\boldsymbol{k}_3 - \frac{845h}{4104}\boldsymbol{k}_4\right).
\end{aligned}$$

The associated six-stage fifth-order method is given by

$$\tilde{\boldsymbol{y}}_{n+1} = \boldsymbol{y}_n + h\left[\frac{16}{135}\boldsymbol{k}_1 + \frac{6656}{12825}\boldsymbol{k}_3 + \frac{28561}{56430}\boldsymbol{k}_4 - \frac{9}{50}\boldsymbol{k}_5 + \frac{2}{55}\boldsymbol{k}_6\right],$$

where $\boldsymbol{k}_1$–$\boldsymbol{k}_5$ are the same as for the fourth-order method above, and

$$\boldsymbol{k}_6 = \boldsymbol{f}\left(t_n + \frac{h}{2}, \boldsymbol{y}_n - \frac{8h}{27}\boldsymbol{k}_1 + 2h\boldsymbol{k}_2 - \frac{3544h}{2565}\boldsymbol{k}_3 + \frac{1859h}{4104}\boldsymbol{k}_4 - \frac{11h}{40}\boldsymbol{k}_5\right).$$

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

$$\kappa = \|\boldsymbol{y}_{n+1} - \tilde{\boldsymbol{y}}_{n+1}\|.$$

The coefficients of the two methods can be listed in a combined Butcher tableaux (see Table 5).

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.

0	0	0	0	0	0	0
$\frac{1}{4}$	$\frac{1}{4}$	0	0	0	0	0
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$	0	0	0	0
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$	0	0	0
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	0	0
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$	0
	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$	0
	$\frac{16}{135}$	0	$\frac{6656}{12825}$	$\frac{28561}{56430}$	$-\frac{9}{50}$	$\frac{2}{55}$

Table 5: Combined Butcher tableaux for the fourth-fifth-order Runge-Kutta-Fehlberg method.

Remark 1. 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 can be found in some textbooks (not ours, though).

2. 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, we saw earlier (in the Matlab script `StiffDemo2.m`) that Runge-Kutta methods become very inefficient.

6 Nonlinear Algebraic Systems

We saw earlier that the implementation of implicit IVP solvers often requires the solution of nonlinear systems of algebraic equations. Nonlinear systems also come up in the solution of boundary values problems of ODEs and PDEs (see later sections). We now briefly discuss some available techniques for the solution of a system of equations

$$G(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{z} = \boldsymbol{0} \tag{35}$$

since essentially any implicit method can be transformed into this form.

Example For the implicit trapezoidal (AM2) method we saw earlier that

$$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{f}(t_n, \boldsymbol{y}_n) + \frac{h}{2}\boldsymbol{f}(t_{n+1}, \boldsymbol{z}).$$

Example If we were to implement the three-step Adams-Moulton formula

$$\boldsymbol{y}_{n+3} = \boldsymbol{y}_{n+2} + \frac{h}{12}\left[5\boldsymbol{f}(t_{n+3}, \boldsymbol{y}_{n+3}) + 8\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}) - \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right]$$

as a stand-alone method (instead of in the context of predictor-corrector methods), then we would have to find a root of $G(\boldsymbol{z}) = \boldsymbol{0}$, where

$$\boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{y}_{n+2} + \frac{h}{12}\left[8\boldsymbol{f}(t_{n+2}, \boldsymbol{y}_{n+2}) - \boldsymbol{f}(t_{n+1}, \boldsymbol{y}_{n+1})\right] + \frac{5h}{12}\boldsymbol{f}(t_{n+3}, \boldsymbol{z}).$$

6.1 Functional Iteration

The functional iteration approach was already discusses in Section 1 in the context of the implicit trapezoidal rule. We simply iterate

$$\boldsymbol{z}^{[i+1]} = \boldsymbol{g}(\boldsymbol{z}^{[i]}), \qquad i = 0, 1, 2, \ldots$$

with a good initial value $\boldsymbol{z}^{[0]}$. As mentioned earlier, convergence is guaranteed by the *Banach fixed-point theorem* provided the norm of the Jacobian of $\boldsymbol{g}$ is small enough, i.e.,

$$\|\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}\| < 1.$$

Remark 1. Functional iteration works rather well as long as the ODE is not stiff. For stiff equations we should use either a Newton-Raphson or modified Newton-Raphson method (see below).

2. Even for non-stiff problems it may happen that functional iteration only converges for very small stepsizes h. Again, it is better to use a Newton-Raphson method.

6.2 Newton-Raphson Iteration

In order to see how Newton-Raphson iteration applies to our standard nonlinear problem (35) we expand about an arbitrary point $\boldsymbol{z}^{[i]}$, i.e.,

$$G(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{z} = \boldsymbol{0} \Longleftrightarrow \boldsymbol{z} = \boldsymbol{g}(\boldsymbol{z}) = \boldsymbol{g}\left(\boldsymbol{z}^{[i]} + (\boldsymbol{z} - \boldsymbol{z}^{[i]})\right)$$

or – using a Taylor expansion –

$$\boldsymbol{z} = \boldsymbol{g}(\boldsymbol{z}^{[i]}) + (\boldsymbol{z} - \boldsymbol{z}^{[i]})\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[i]}) + \mathcal{O}\left(\|\boldsymbol{z} - \boldsymbol{z}^{[i]}\|^2\right).$$

If we now proceed as in the derivation of the standard Newton method and drop the second-order terms then we end up with

$$\boldsymbol{z} - \boldsymbol{z}^{[i]} \approx \boldsymbol{g}(\boldsymbol{z}^{[i]}) + (\boldsymbol{z} - \boldsymbol{z}^{[i]})\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[i]}) - \boldsymbol{z}^{[i]}$$

or

$$\left(I - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[i]})\right)(\boldsymbol{z} - \boldsymbol{z}^{[i]}) \approx \boldsymbol{g}(\boldsymbol{z}^{[i]}) - \boldsymbol{z}^{[i]}.$$

This now motivates the iterative method

$$\boldsymbol{z}^{[i+1]} = \boldsymbol{z}^{[i]} - \left(I - \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[i]})\right)^{-1}\left[\boldsymbol{z}^{[i]} - \boldsymbol{g}(\boldsymbol{z}^{[i]})\right], \qquad i = 0, 1, \ldots, \tag{36}$$

which is known as the *Newton-Raphson* method.

Note the similarity with the standard (scalar) Newton method

$$z^{[i+1]} = z^{[i]} - \frac{g(z^{[i]}) - z^{[i]}}{g'(z^{[i]}) - 1}, \qquad i = 0, 1, \ldots,$$

which corresponds to (using $G(z) = g(z) - z$)

$$z^{[i+1]} = z^{[i]} - \frac{G(z^{[i]})}{G'(z^{[i]})}, \qquad i = 0, 1, \ldots,$$

– the well-known formula for finding a zero of G.

Remark This is essentially the same as in the textbook. There, however, the function $\boldsymbol{g}$ is represented in the form $h\boldsymbol{g}(\boldsymbol{z}) + \boldsymbol{\beta}$.

As with the standard Newton iteration, one can show that – for a "good" starting value $\boldsymbol{z}^{[0]}$ – the iteration converges *quadratically*, i.e., the error satisfies

$$\|\boldsymbol{z} - \boldsymbol{z}^{[i+1]}\| \leq c\|\boldsymbol{z} - \boldsymbol{z}^{[i]}\|^2,$$

where $\boldsymbol{z}$ is the exact root of $G(\boldsymbol{z}) = \boldsymbol{g}(\boldsymbol{z}) - \boldsymbol{z}$. In particular, if G is linear, then Newton-Raphson iteration converges in a single step (cf. the linear boundary value problems in the next section).

Example Solve

$$\begin{aligned} x^2 + y^2 &= 4 \\ xy &= 1 \end{aligned}$$

which corresponds to finding the intersection points of a circle and a hyperbola in the plane. Here

$$G(\boldsymbol{z}) = G(x, y) = \begin{bmatrix} G_1(x, y) \\ G_2(x, y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 4 \\ xy - 1 \end{bmatrix}$$

and

$$\frac{\partial G}{\partial \boldsymbol{z}} = J(x, y) = \begin{bmatrix} \frac{\partial G_1}{\partial x} & \frac{\partial G_1}{\partial y} \\ \frac{\partial G_2}{\partial x} & \frac{\partial G_2}{\partial y} \end{bmatrix} (x, y) = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}.$$

This example is illustrated in the Matlab script `run_newtonmv.m`.

Remark 1. It is apparent from formula (36) that implementation of the Newton-Raphson method is rather expensive since we need to compute and evaluate the entire Jacobian matrix $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[i]})$ at every iteration.

2. Moreover, each Newton-Raphson iteration requires the *inverse* of the Jacobian. This, however, corresponds to the solution of a system of linear equations – another expensive task.

In order to avoid both of these heavy computational burdens a *modified Newton-Raphson* method has been proposed.

6.3 Modified Newton-Raphson Iteration

In the *modified Newton-Raphson* method we approximate the Jacobian (which really should change in each iteration, cf. (36) by a *fixed matrix*, e.g., $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}}(\boldsymbol{z}^{[0]})$. Now, we no longer need to re-compute the Jacobian in every iteration, nor do we need to solve the linear system (invert the Jacobian) in every iteration. However, quadratic convergence is lost.

Remark Other, so-called *quasi-Newton methods* exist, that are analogous to the scalar *secant method* (and thus avoid computation of the jacobian). These methods are more cheaper to use than Newton-Raphson iteration. One such method is *Broyden's method*. It can be found it some textbooks on numerical analysis, e.g., Kincaid and Cheney's "Numerical Analysis: Mathematics of Scientific Computing". A discussion of that method goes beyond the scope of this course.

7 Boundary Value Problems for ODEs

Boundary values for ODEs are not covered in the textbook. We discuss this important subject in the scalar case (single equation) only.

7.1 Boundary Value Problems: Theory

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

$$\begin{gathered} y''(t) = f(t, y(t), y'(t)) \\ a_0 y(a) + a_1 y'(a) = \alpha, \quad b_0 y(b) + b_1 y'(b) = \beta. \end{gathered} \tag{37}$$

Remark 1. Note that this kind of problem can no longer be converted to a system of two first order initial value problems as we have been doing thus far.

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 (37).

Theorem 7.1 *Suppose f in (37) 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. *$a_0 a_1 \leq 0$, $b_0 b_1 \geq 0$, and $|a_0| + |b_0| > 0$, $|a_0| + |a_1| > 0$, $|b_0| + |b_1| > 0$,*

then the boundary value problem (37) has a unique solution.

Example Consider the BVP

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

To apply Theorem 7.1 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 (37) takes the special form

$$\begin{aligned} y''(t) &= u(t) + v(t)y(t) + w(t)y'(t) \\ y(a) &= \alpha, \quad y(b) = \beta, \end{aligned} \tag{38}$$

then it is called *linear*. In this case Theorem 7.1 simplifies considerably.

Theorem 7.2 *If u, v, w in (38) are continuous and $v(t) > 0$ on $[a, b]$, then the linear boundary value problem (38) has a unique solution.*

Remark A classical reference for the numerical solution of two-point BVPs is the book "Numerical Methods for Two-Point Boundary Value Problems" by H. B. Keller (1968). A modern reference is "Numerical Solution of Boundary Value Problems for Ordinary Differential Equations" by Ascher, Mattheij, and Russell (1995).

7.2 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

$$\begin{aligned} y''(t) &= f(t, y(t), y'(t)) \\ y(a) &= \alpha, \quad y(b) = \beta. \end{aligned} \tag{39}$$

With some modifications the methods discussed below can also be applied to the more general problem (37).

The fundamental idea on which the so-called *shooting methods* are based is to formulate an initial value problem associated with (39). Namely,

$$\begin{aligned} y''(t) &= f(t, y(t), y'(t)) \\ y(a) &= \alpha, \quad y'(a) = z. \end{aligned} \tag{40}$$

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 s-step methods), and thus obtain a solution y_z. In order to see how well this solution matches the solution y of the two-point boundary value problem (39) we compute the difference

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

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 systems of equations (such as functional iteration or Newton-Raphson iteration discussed in the previous section) to find a better slope.

Remark 1. Changing the "aim" of the initial value problem by adjusting the initial slope to "hit" the target value $y(b) = \beta$ 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

$$\begin{aligned} y_1'(t) &= \lambda y_1(t) \\ y_2'(t) &= -\lambda y_2(t) \\ y_1(0) = 1, &\quad y_2(a) = 1 \end{aligned}$$

for $t \in [0, a]$ is stable since the solution $y_1(t) = e^{\lambda t}$, $y_2(t) = e^{a\lambda}e^{-\lambda t}$ remains bounded for $t \to \infty$ even for large values of a. On the other hand, the initial value problem

$$\begin{aligned} y_1'(t) &= \lambda y_1(t) \\ y_2'(t) &= -\lambda y_2(t) \\ y_1(0) = \alpha, &\quad y_2(0) = \beta \end{aligned}$$

is unstable for any $\lambda \neq 0$ since always one of the components of the solution $y_1(t) = \alpha e^{\lambda t}$, $y_2(t) = \beta e^{-\lambda t}$ 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.

We now explain how we can use Newton iteration as part of the shooting method. Newton's method for solving the nonlinear equation $\phi(z) = 0$ is

$$z^{[i+1]} = z^{[i]} - \frac{\phi(z^{[i]})}{\phi'(z^{[i]})}, \quad i \geq 0.$$

Now the problem is to obtain the value $\phi'(z^{[i]})$. Note that this is anything but obvious, since we do not even have an expression for the function ϕ – only for the value $\phi(z^{[i]})$.

In order to obtain an expression for $\phi'(z^{[i]})$ we consider the initial value problem (40) in the form

$$\begin{aligned} y''(t,z) &= f(t, y(t,z), y'(t,z)) \\ y(a,z) = \alpha, &\quad y'(a,z) = z. \end{aligned} \tag{41}$$

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

$$\begin{aligned} \frac{\partial y''(t,z)}{\partial z} &= \frac{\partial}{\partial z} f(t, y(t,z), y'(t,z)) \\ &= \frac{\partial f}{\partial y}\frac{\partial y}{\partial z} + \frac{\partial f}{\partial y'}\frac{\partial y'}{\partial z}, \end{aligned} \tag{42}$$

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

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

If we introduce the notation $v(t) = \frac{\partial y}{\partial z}(t,z)$, then (42) becomes

$$\begin{aligned} v''(t) = \tfrac{\partial f}{\partial y}(t,y(t),y'(t))v(t) + \tfrac{\partial f}{\partial y'}(t,y(t),y'(t))v'(t) \\ v(a) = 0, \quad v'(a) = 1. \end{aligned} \tag{43}$$

Equation (43) is called the *first variational equation*. We can recognize this as another initial value problem for the function v.

Now,

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

so that

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

Therefore, we can obtain the value $\phi'(z_k)$ required in Newton's method by solving the initial value problem (43) up to $t = b$.

Algorithm

1. Provide an initial guess z_0 and a tolerance δ.

2. Solve the initial value problems (40) and (43) with initial conditions

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

   respectively. Let $i = 0$. This provides us with $\phi(z^{[i]}) = y_{z^{[i]}}(b) - \beta$ and $\phi'(z^{[i]}) = v(b)$.

3. Apply Newton's method, i.e., compute

   $$z^{[i+1]} = z^{[i]} - \frac{\phi(z^{[i]})}{\phi'(z^{[i]})}.$$

4. Check if $|\phi(z^{[i+1]})| < \delta$. If yes, stop. Otherwise, increment i and repeat from 3.

Remark 1. Note that the computation of $\phi(z^{[i+1]})$ in Step 4 requires solution of an IVP (40).

2. The initial value problems (40) and (43) can be solved simultaneously using a vectorized IVP solver.

3. If the boundary value problem (39) is linear, then the function ϕ will also be linear, and therefore a single step of the Newton method will provide the "correct" initial slope.

4. 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* (even in the scalar case!) which can then be solved using a (modified) Newton-Raphson method. This approach is known as a *multiple shooting method*. More details can be found in the book "Introduction to Numerical Analysis" by Stoer and Bulirsch (1980).

7.3 Boundary Value Problems: Finite Differences

Again we consider the boundary value problem

$$\begin{aligned} y''(t) &= f(t, y(t), y'(t)) \\ y(a) &= \alpha, \quad y(b) = \beta. \end{aligned} \tag{44}$$

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

$$t_k = a + kh, \ k = 0, 1, \ldots, m+1, \quad \text{and} \quad h = \frac{b-a}{m+1}.$$

The basic idea is to *discretize* the differential equation (44) on the given partition.

Before we attempt to solve the BVP (44) we first review approximation of (continuous) derivatives by (discrete) differences.

7.3.1 Numerical Differentiation

From calculus we know that the value of the derivative of a given function f at some point x in its domain can be approximated via

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{45}$$

where h is small. In order to get an error estimate for this approximation we use a Taylor expansion of f

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\eta), \qquad \eta \in (x, x+h).$$

This implies

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\eta),$$

i.e., the truncation error for the standard difference approximation of the first derivative is $\mathcal{O}(h)$.

We now consider a more accurate approximation. To this end we take *two* Taylor expansions

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(\eta_1), \tag{46}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{6} f'''(\eta_2). \tag{47}$$

Subtracting (47) from (46) yields

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}\left[f'''(\eta_1) + f'''(\eta_2)\right]$$

or the following formula

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \mathcal{O}(h^2) \tag{48}$$

for the first derivative which is more accurate than (45).

Similarly, by adding (46) and (47) we obtain the following formula for the second derivative

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + \mathcal{O}(h^2). \tag{49}$$

As with the basic numerical integration methods, there is again a close connection between numerical differentiation methods and polynomial interpolation. If we have information of f at $\nu+1$ points, then we can find an interpolating polynomial p of degree ν. We then differentiate p to get an estimate for the derivative of f.

Consider the error formula for the Lagrange form of the interpolating polynomial (see (1) in Chapter 1)

$$f(x) - p(x) = \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta_x) w(x)$$

or

$$f(x) = \sum_{k=0}^{\nu} f(\xi_k) p_k(x) + \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta_x) w(x)$$

where $w(x) = \prod_{k=0}^{\nu} (x - \xi_k)$, and the p_k are the Lagrange basis polynomials as studied in Chapter 1. It is important for the next step to note that the point η in the error formula depends on the evaluation point x. This explains the use of the notation η_x.

Differentiation then leads to

$$f'(x) = \sum_{k=0}^{\nu} f(\xi_k) p_k'(x) + \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta_x) w'(x) + \frac{1}{(\nu+1)!} \frac{d}{dx}\left[f^{(\nu+1)}(\eta_x)\right] w(x).$$

Let us now assume that the evaluation point x is located at one of the interpolation nodes, ξ_j say, i.e., we know f at certain points, and want to estimate f' at (some of) those same points. Then $w(\xi_j) = 0$ and

$$f'(\xi_j) = \sum_{k=0}^{\nu} f(\xi_k) p_k'(\xi_j) + \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta_{\xi_j}) w'(\xi_j).$$

One can simplify this expression to

$$f'(\xi_j) = \sum_{k=0}^{\nu} f(\xi_k) p_k'(\xi_j) + \frac{1}{(\nu+1)!} f^{(\nu+1)}(\eta_{\xi_j}) \prod_{\substack{i=0 \\ i \neq j}}^{\nu} (\xi_j - \xi_i). \tag{50}$$

Remark 1. If all nodes are equally spaced with spacing h, then (50) in an $\mathcal{O}(h^\nu)$ formula.

2. The values $p_k'(\xi_j)$ in (50) are called the *coefficients* of the derivative formula.

Example 1. Using linear interpolation at two equally spaced points, $\xi_0 = x$ and $\xi_1 = x + h$, leads to the estimate (45).

2. (48) is obtained by performing quadratic interpolation at $\xi_0 = x - h$, $\xi_1 = x$, and $\xi_2 = x + h$.

3. (49) is obtained by performing quadratic interpolation at $\xi_0 = x - h$, $\xi_1 = x$, and $\xi_2 = x + h$.

4. These and other examples are illustrated in the Maple worksheet `472_DerivativeEstimates.mws`.

Remark The discussion in Section 7.1 of the textbook employs a more abstract framework based on discrete *finite difference operators* and formal Taylor expansions of these operators.

We now return the the finite difference approximation of the BVP (44) and introduce the following formulas for the first and second derivatives:

$$\begin{aligned} y'(t) &= \frac{y(t+h) - y(t-h)}{2h} - \frac{h^2}{6} y^{(3)}(\eta) \\ y''(t) &= \frac{y(t+h) - 2y(t) + y(t-h)}{h^2} - \frac{h^2}{12} y^{(4)}(\tau). \end{aligned} \tag{51}$$

If we use the notation $y_k = y(t_k)$ along with the finite difference approximations (51), then the boundary value problem (44) becomes

$$\begin{aligned} y_0 &= \alpha \\ \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} &= f\left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h}\right), \quad k = 1, \ldots, m, \\ y_{m+1} &= \beta. \end{aligned} \tag{52}$$

7.3.2 Linear Finite Differences

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

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

Then (52) becomes

$$\begin{aligned} y_0 &= \alpha \\ \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} &= u_k + v_k y_k + w_k \frac{y_{k+1} - y_{k-1}}{2h}, \quad k = 1, \ldots, m, \\ y_{m+1} &= \beta, \end{aligned} \tag{53}$$

where we have used the notation $u_k = u(t_k)$, $v_k = v(t_k)$, and $w_k = w(t_k)$. This is a system of m linear equations for the m unknowns y_k, $k = 1, \ldots, m$. In fact, the system is *tridiagonal*. This can be seen if we rewrite (53) as

$$y_0 = \alpha$$

$$\begin{aligned}\left(-1-\frac{w_k}{2}h\right)y_{k-1}+\left(2+h^2v_k\right)y_k+\left(-1+\frac{w_k}{2}h\right)y_{k+1} &= -h^2u_k, \quad k=1,\ldots,m,\\ y_{m+1} &= \beta,\end{aligned}$$

or in matrix form

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

$$\times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{m-1} \\ y_m \end{bmatrix} = \begin{bmatrix} -h^2u_1-\alpha\left(-1-\frac{w_1}{2}h\right) \\ -h^2u_2 \\ \vdots \\ -h^2u_{m-1} \\ -h^2u_m-\beta\left(-1+\frac{w_m}{2}h\right) \end{bmatrix}.$$

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 $\{y_k \ : \ k=0,1,\ldots,m+1\}$. 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

$$|2+h^2v_k| > |1+\frac{h}{2}w_k|+|1-\frac{h}{2}w_k|.$$

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

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

which is obviously true.

Remark

1. The assumption $v_k>0$ is no real restriction since this is also a condition for the Existence and Uniqueness Theorem 7.2.
2. The assumption $|\frac{h}{2}w_k|<1$ on the mesh size h is a little more difficult to verify.

For the linear finite difference method one can give error bounds.

Theorem 7.3 *The maximum pointwise error of the linear finite difference method is given by*

$$\max_{k=1,\ldots,m}|y(t_k)-y_k| \le Ch^2, \quad \text{as } h\to 0,$$

where $y(t_k)$ is the exact solution at t_k, and y_k is the corresponding approximate solution obtained by the finite difference method.

Proof For the exact solution we have for any $k = 1, \ldots, m$

$$y''(t_k) = u(t_k) + v(t_k)y(t_k) + w(t_k)y'(t_k)$$

or

$$\frac{y(t_k+h) - 2y(t_k) + y(t_k-h)}{h^2} - \frac{h^2}{12}y^{(4)}(\tau_k) = u_k + v_k y(t_k) + w_k \left[\frac{y(t_k+h) - y(t_k-h)}{2h} - \frac{h^2}{6}y^{(3)}(\eta_k)\right], \tag{54}$$

whereas for the approximate solution we have the relation

$$\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} = u_k + v_k y_k + w_k \frac{y_{k+1} - y_{k-1}}{2h}$$

(cf. (53)). Subtracting (53) from equation (54) yields

$$\frac{e_{k+1} - 2e_k + e_{k-1}}{h^2} = v_k e_k + w_k \frac{e_{k+1} - e_k}{2h} + h^2 g_k, \tag{55}$$

where

$$e_k = y(t_k) - y_k$$

and

$$g_k = \frac{1}{12}y^{(4)}(\tau_k) - \frac{1}{6}y^{(3)}(\eta_k).$$

Since (55) is analogous to (53) it can be rewritten as

$$\left(-1 - \frac{w_k}{2}h\right)e_{k-1} + \left(2 + h^2 v_k\right)e_k + \left(-1 + \frac{w_k}{2}h\right)e_{k+1} = -h^4 g_k.$$

Then we get

$$\left|\left(2 + h^2 v_k\right)e_k\right| = \left|-\left(-1 - \frac{w_k}{2}h\right)e_{k-1} - \left(-1 + \frac{w_k}{2}h\right)e_{k+1} - h^4 g_k\right|$$

and using the triangle inequality

$$\left|\left(2 + h^2 v_k\right)e_k\right| \le \left|\left(-1 - \frac{w_k}{2}h\right)e_{k-1}\right| + \left|\left(-1 + \frac{w_k}{2}h\right)e_{k+1}\right| + \left|h^4 g_k\right|.$$

Now we let $\lambda = \|e\|_\infty = \max_{j=1,\ldots,m} |e_j|$, and pick the index k such that

$$|e_k| = \|e\|_\infty = \lambda,$$

i.e., we look at the largest of the errors. Therefore

$$|2 + h^2 v_k| \underbrace{|e_k|}_{=\lambda} \le h^4 |g_k| + |-1 + \frac{w_k}{2}h| \underbrace{|e_{k+1}|}_{\le\lambda} + |-1 - \frac{w_k}{2}h| \underbrace{|e_{k-1}|}_{\le\lambda}.$$

Using the definition of λ, and bounding $|g_k|$ by its maximum we have

$$\lambda\left(|2 + h^2 v_k| - |-1 + \frac{w_k}{2}h| - |-1 - \frac{w_k}{2}h|\right) \le h^4 \|g\|_\infty.$$

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

$$\lambda h^2 v_k \leq h^4 \|g\|_\infty \quad \Longleftrightarrow \quad \lambda v_k \leq h^2 \|g\|_\infty,$$

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

$$\max_{k=1,\ldots,m} |y(t_k) - y_k| \leq Ch^2,$$

where

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

■

Remark The error bound in Theorem 7.3 holds only for C^4 functions y, whereas for the solution to exist only C^2 continuity is required.

7.3.3 Nonlinear Finite Differences

We now return to the original discretization

$$\begin{aligned} y_0 &= \alpha \\ \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} &= f\left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h}\right), \quad k = 1, \ldots, m, \\ y_{m+1} &= \beta \end{aligned}$$

of the boundary value problem (44). However, now we allow f to be a nonlinear function. This leads to the following system of *nonlinear* equations:

$$\begin{aligned} 2y_1 - y_2 + h^2 f\left(t_1, y_1, \frac{y_2 - \alpha}{2h}\right) - \alpha &= 0 \\ -y_{k-1} + 2y_k - y_{k+1} + h^2 f\left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h}\right) &= 0, \quad k = 2, \ldots, m-1, \\ -y_{m-1} + 2y_m + h^2 f\left(t_m, y_m, \frac{\beta - y_{m-1}}{2h}\right) - \beta &= 0. \end{aligned} \tag{56}$$

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 7.1.

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

$$J(\mathbf{y}^{[i]})\mathbf{u} = -F(\mathbf{y}^{[i]})$$

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

$$\mathbf{y}^{[i+1]} = \mathbf{y}^{[i]} + \mathbf{u},$$

where $\mathbf{y}^{[i]}$ is the i-th iterate of the vector of grid values $y_0, y_1, \ldots, y_{m+1}$, and J is the tridiagonal Jacobian matrix defined by

$$J(\mathbf{y})_{k\ell} = \begin{cases} -1 + \frac{h}{2} f_z \left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h} \right), & k = \ell - 1, \ \ell = 2, \ldots, m, \\ 2 + h^2 f_y \left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h} \right), & k = \ell, \ \ell = 1, \ldots, m, \\ -1 - \frac{h}{2} f_z \left(t_k, y_k, \frac{y_{k+1} - y_{k-1}}{2h} \right), & k = \ell + 1, \ \ell = 1, \ldots, m - 1. \end{cases}$$

Here $f = f(t, y, z)$ and $F(\mathbf{y})$ is given by the left-hand side of the equations in (56).

Remark 1. As always, Newton iteration requires a "good" initial guess $y_1, \ldots, y_n$.

2. One can show that the nonlinear finite difference method also has $\mathcal{O}(h^2)$ convergence order.

8 Boundary Value Problems for PDEs

Before we specialize to boundary value problems for PDEs – which only make sense for elliptic equations – we need to explain the terminology "elliptic".

8.1 Classification of Partial Differential Equations

We therefore consider general second-order partial differential equations (PDEs) of the form

$$Lu = au_{tt} + bu_{xt} + cu_{xx} + f = 0, \tag{57}$$

where u is an unknown function of x and t, and a, b, c, and f are given functions. If these functions depend only on x and t, then the PDE (57) is called *linear*. If a, b, c, or f depend also on u, u_x, or u_t, then the PDE is called *quasi-linear*.

Remark 1. The notation used in (57) suggests that we think of one of the variables, t, as time, and the other, x, as space.

2. In principle, we could also have second-order PDEs involving more than one space dimension. However, we limit the discussion here to PDEs with a total of two independent variables.

3. Of course, a second-order PDE can also be independent of time, and contain two space variables only (such as Laplace's equation). These will be the elliptic equations we are primarily interested in.

There are three fundamentally different types of second-order quasi-linear PDEs:

- If $b^2 - 4ac > 0$, then L is *hyperbolic*.
- If $b^2 - 4ac = 0$, then L is *parabolic*.
- If $b^2 - 4ac < 0$, then L is *elliptic*.

Example 1. The *wave equation*

$$u_{tt} = \alpha^2 u_{xx} + f(x,t)$$

is a second-order linear hyperbolic PDE since $a \equiv 1$, $b \equiv 0$, and $c \equiv -\alpha^2$, so that

$$b^2 - 4ac = 4\alpha^2 > 0.$$

2. The *heat* or *diffusion equation*

$$u_t = ku_{xx}$$

is a second-order quasi-linear parabolic PDE since $a = b \equiv 0$, and $c \equiv -k$, so that

$$b^2 - 4ac = 0.$$

3. For *Poisson's equation* (or *Laplace's equation* in case $f \equiv 0$)

$$u_{xx} + u_{yy} = f(x, y)$$

we use y instead of t. This is a second-order linear elliptic PDE since $a = c \equiv 1$ and $b \equiv 0$, so that

$$b^2 - 4ac = -4 < 0.$$

Remark Since a, b, and c may depend on x, t, u, u_x, and u_t the classification of the PDE may even vary from point to point.

8.2 Boundary Value Problems for Elliptic PDEs: Finite Differences

We now consider a boundary value problem for an elliptic partial differential equation. The discussion here is similar to Section 7.2 in the textbook.

We use the following Poisson equation in the unit square as our model problem, i.e.,

$$\begin{aligned} \nabla^2 u = u_{xx} + u_{yy} &= f(x, y), \qquad (x, y) \in \Omega = (0, 1)^2, \\ u(x, y) &= \phi(x, y), \qquad (x, y) \text{ on } \partial\Omega. \end{aligned} \tag{58}$$

This problem arises, e.g., when we want to determine the steady-state temperature distribution u in a square region with prescribed boundary temperature ϕ. Of course, this simple problem can be solved analytically using Fourier series.

However, we are interested in numerical methods. Therefore, in this section, we use the usual finite difference discretization of the partial derivatives, i.e.,

$$u_{xx}(x, y) = \frac{1}{h^2} \left[u(x + h, y) - 2u(x, y) + u(x - h, y) \right] + \mathcal{O}(h^2) \tag{59}$$

and

$$u_{yy}(x, y) = \frac{1}{h^2} \left[u(x, y + h) - 2u(x, y) + u(x, y - h) \right] + \mathcal{O}(h^2). \tag{60}$$

The computational grid introduced in the domain $\overline{\Omega} = [0, 1]^2$ is now

$$(x_k, y_\ell) = (kh, \ell h), \quad k, \ell = 0, \ldots, m + 1,$$

with *mesh size* $h = \frac{1}{m+1}$.

Using the compact notation

$$u_{k,\ell} = u(x_k, y_\ell), \quad u_{k+1,\ell} = u(x_k + h, y_\ell), \quad \text{etc.},$$

The Poisson equation (58) turns into the difference equation

$$\frac{1}{h^2} \left[u_{k-1,\ell} - 2u_{k,\ell} + u_{k+1,\ell} \right] + \frac{1}{h^2} \left[u_{k,\ell-1} - 2u_{k,\ell} + u_{k,\ell+1} \right] = f_{k,\ell}. \tag{61}$$

This equation can be rewritten as

$$4u_{k,\ell} - u_{k-1,\ell} - u_{k+1,\ell} - u_{k,\ell-1} - u_{k,\ell+1} = -h^2 f_{k,\ell}. \tag{62}$$

Example Let's consider a computational mesh of 5×5 points, i.e., $h = \frac{1}{4}$, or $m = 3$. Discretizing the boundary conditions in (58), the values of the approximate solution around the boundary

$$\begin{array}{ll} u_{0,\ell},\ u_{4,\ell} & \ell = 0, \ldots, 4, \\ u_{u,0},\ u_{k,4} & k = 0, \ldots, 4, \end{array}$$

are determined by the appropriate values of ϕ. There remain 9 points in the interior of the domain that have to be determined using the *stencil* (62). Figure 1 illustrates one instance of this task. By applying the stencil to each of the interior points, we obtain 9 conditions for the 9 undetermined values.

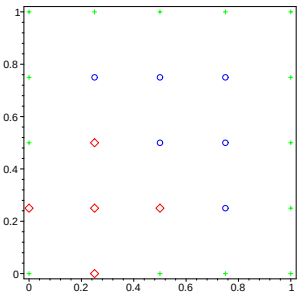

Figure 1: Illustration of finite difference method for Poisson equation on 5×5 grid. Interior mesh points are indicated with blue $\circ$, green $+$ correspond to given boundary values, and points marked with red $\diamondsuit$ form a typical stencil.

Thus, we obtain the following 9 equations

$$\begin{aligned} 4u_{1,1} - u_{2,1} - u_{1,2} &= u_{0,1} + u_{1,0} - h^2 f_{1,1} \\ 4u_{2,1} - u_{1,1} - u_{3,1} - u_{2,2} &= u_{2,0} - h^2 f_{2,1} \\ 4u_{3,1} - u_{2,1} - u_{3,2} &= u_{4,1} + u_{3,0} - h^2 f_{3,1} \\ 4u_{1,2} - u_{2,2} - u_{1,1} - u_{1,3} &= u_{0,2} - h^2 f_{1,2} \\ 4u_{2,2} - u_{1,2} - u_{3,2} - u_{2,1} - u_{2,3} &= -h^2 f_{2,2} \\ 4u_{3,2} - u_{2,2} - u_{3,1} - u_{3,3} &= u_{4,2} - h^2 f_{3,2} \\ 4u_{1,3} - u_{2,3} - u_{1,2} &= u_{1,4} + u_{0,3} - h^2 f_{1,3} \\ 4u_{2,3} - u_{1,3} - u_{3,3} - u_{2,2} &= u_{2,4} - h^2 f_{2,3} \\ 4u_{3,3} - u_{2,3} - u_{3,2} &= u_{4,3} + u_{3,4} - h^2 f_{3,3}. \end{aligned}$$

The first equation corresponds to the stencil shown in Figure 1. The other equations are obtained by moving the stencil row-by-row across the grid from left to right.

We can also write the above equations in matrix form. To this end we introduce the vector

$$\boldsymbol{u} = [u_{1,1}, u_{2,1}, u_{3,1}, u_{1,2}, u_{2,2}, u_{3,2}, u_{1,3}, u_{2,3}, u_{3,3}]^T$$

of unknowns. Here we have used the *natural* (row-by-row) ordering of the mesh points. Then we get

$$A\boldsymbol{u} = \boldsymbol{b}$$

with

$$A = \begin{bmatrix} \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} & \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \\ \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} & \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} & \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} & \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \end{bmatrix}$$

and

$$\boldsymbol{b} = \begin{bmatrix} u_{0,1} + u_{1,0} - h^2 f_{1,1} \\ u_{2,0} - h^2 f_{2,1} \\ u_{4,1} + u_{3,0} - h^2 f_{3,1} \\ u_{0,2} - h^2 f_{1,2} \\ -h^2 f_{2,2} \\ u_{4,2} - h^2 f_{3,2} \\ u_{1,4} + u_{0,3} - h^2 f_{1,3} \\ u_{2,4} - h^2 f_{2,3} \\ u_{4,3} + u_{3,4} - h^2 f_{3,3} \end{bmatrix}.$$

We can see that A is a block-tridiagonal matrix of the form

$$A = \begin{bmatrix} T & -I & O \\ -I & T & -I \\ O & -I & T \end{bmatrix}.$$

In general, for problems with $m \times m$ interior mesh points, A will be of size $m^2 \times m^2$ (since there are m^2 unknown values at interior mesh points), but contain no more than $5m^2$ nonzero entries (since equation (62) involves at most 5 points at one time). Thus, A is a classical example of a *sparse matrix*. Moreover, A still has a block-tridiagonal structure

$$A = \begin{bmatrix} T & -I & O & \dots & O \\ -I & T & -I & & \vdots \\ O & \ddots & \ddots & \ddots & O \\ \vdots & & -I & T & -I \\ O & \dots & O & -I & T \end{bmatrix}$$

with $m \times m$ blocks

$$T = \begin{bmatrix} 4 & -1 & 0 & \dots & 0 \\ -1 & 4 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 4 & -1 \\ 0 & \dots & 0 & -1 & 4 \end{bmatrix}$$

as well as $m \times m$ identity matrices I, and zero matrices O.

Remark 1. Since A is sparse (and symmetric positive definite) it lends itself to an application of an iterative system solver such as *Gauss-Seidel iteration*. After initializing the values at all mesh points (including those along the boundary) to some appropriate value (in many cases zero will work), we can simply iterate with formula (62), i.e., we obtain the algorithm fragment for M steps of Gauss-Seidel iteration

for $i = 1$ to M do

for $k = 1$ to m do

for $\ell = 1$ to m do

$$u_{k,\ell} = \left(u_{k-1,\ell} + u_{k+1,\ell} + u_{k,\ell-1} + u_{k,\ell+1} - h^2 f_{k,\ell}\right)/4$$

end

end

end

Note that the matrix A never has to be fully formed or stored during the computation.

2. State-of-the-art algorithms for the Poisson (or homogeneous Laplace) equation are so-called *fast Poisson solvers* based on the fast Fourier transform, or multigrid methods.

While we know that at each gridpoint the Laplacian $u_{xx} + u_{yy}$ is approximated by finite differences with accuracy $\mathcal{O}(h^2)$, one can show that (globally) the error is also of order $\mathcal{O}(h^2)$.

Theorem 8.1 *The maximum pointwise error of the finite difference method with the 5-point stencil introduced above applied to the Poisson problem on a square, rectangular, or L-shaped domain is given by*

$$\max_{k,\ell=1,\ldots,m} |u(x_k, y_\ell) - u_{k,\ell}| \leq Ch^2, \quad \text{as } h \to 0,$$

where $u(x_k, y_\ell)$ is the exact solution at (x_k, y_ℓ), and $u_{k,\ell}$ is the corresponding approximate solution obtained by the finite difference method.

We emphasize that this estimate holds only for the type of domains specified in the theorem. If the stencil does not match the domain exactly, then we need to use special boundary correction terms to maintain $\mathcal{O}(h^2)$ accuracy (more details are given in the textbook on pages 121/122).

9 Boundary Value Problems: Collocation

We now present a different type of numerical method that will yield the approximate solution of a boundary value problem in the form of a function, as opposed to the set of discrete points resulting from the methods studied earlier. Just like the finite difference method, this method applies to both one-dimensional (two-point) boundary value problems, as well as to higher-dimensional elliptic problems (such as the Poisson problem).

We limit our discussion to the one-dimensional case. Assume we are given a general linear two-point boundary value problem of the form

$$\begin{aligned} Ly(t) &= f(t), \qquad t \in [a,b], \\ y(a) &= \alpha, \quad y(b) = \beta. \end{aligned} \tag{63}$$

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

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

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

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

and (63) becomes

$$\sum_{j=1}^{n} c_j L v_j(t) = f(t), \qquad t \in [a,b], \tag{64}$$

$$\sum_{j=1}^{n} c_j v_j(a) = \alpha, \quad \sum_{j=1}^{n} c_j v_j(b) = \beta. \tag{65}$$

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 (64) 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* $t_2, \ldots, t_{n-1}$, 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 $t_1 = a$ and $t_n = b$, then (64) becomes

$$\begin{aligned}
\sum_{j=1}^{n} c_j v_j(t_1) &= \alpha, \\
\sum_{j=1}^{n} c_j L v_j(t_i) &= f(t_i), \qquad i = 2, \ldots, n-1, \\
\sum_{j=1}^{n} c_j v_j(t_n) &= \beta.
\end{aligned}$$

In matrix form we have the linear system

$$\begin{bmatrix}
v_1(t_1) & v_2(t_1) & \ldots & v_n(t_1) \\
Lv_1(t_2) & Lv_2(t_2) & \ldots & Lv_n(t_2) \\
\vdots & & & \vdots \\
Lv_1(t_{n-1}) & Lv_2(t_{n-1}) & \ldots & Lv_n(t_{n-1}) \\
v_1(t_n) & v_2(t_n) & \ldots & v_n(t_n)
\end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}
=
\begin{bmatrix} \alpha \\ f(t_2) \\ \vdots \\ f(t_{n-1}) \\ \beta \end{bmatrix}. \tag{66}$$

If the space V and the collocation points t_i, $i = 1, \ldots, n$, are chosen such that the collocation matrix in (66) is nonsingular then we can represent an approximate solution of (63) from the space V uniquely as

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

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.

9.1 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 $r = |\cdot - t|$ the distance from a fixed center t. If we center a one multiquadric at each of the collocation points t_j, $j = 1, \ldots, n$, then the approximation space becomes

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

Now the system (66) becomes

$$\begin{bmatrix}
\phi(|t_1 - t_1|) & \phi(|t_1 - t_2|) & \ldots & \phi(|t_1 - t_n|) \\
L\phi(|t_2 - t_1|) & L\phi(|t_2 - t_2|) & \ldots & L\phi(|t_2 - t_n|) \\
\vdots & & & \vdots \\
L\phi(|t_{n-1} - t_1|) & L\phi(|t_{n-1} - t_2|) & \ldots & L\phi(|t_{n-1} - t_n|) \\
\phi(|t_n - t_1|) & \phi(|t_n - t_2|) & \ldots & \phi(|t_n - t_n|)
\end{bmatrix}
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}
=
\begin{bmatrix} \alpha \\ f(t_2) \\ \vdots \\ f(t_{n-1}) \\ \beta \end{bmatrix}. \tag{67}$$

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

Example Let the differential operator L be given by

$$Ly(t) = y''(t) + wy'(t) + vy(t),$$

and ϕ denote the multiquadric radial basic function. Then

$$L\phi(|t-\tau|) = \phi''(|t-\tau|) + w\phi'(|t-\tau|) + v\phi(|t-\tau|)$$

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

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

and the collocation matrix has entries of this type in rows 2 through $n-1$ with $\tau = t_j$, $j = 2, \ldots, n-1$.

Remark 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 for an arbitrary choice of collocation points. 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 (67) 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

$$V = \text{span}\{\phi(|\cdot - t_1|), \phi(|\cdot - t_n|)\} \cup \text{span}\{L^{(2)}\phi(|\cdot - t_j|),\ j = 2, \ldots, n-1\}.$$

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

Since the approximate solution is now of the form

$$y(t) = c_1\phi(|t - t_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|t - t_j|) + c_n\phi(|t - t_n|) \tag{68}$$

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

We start with (64), which – based on (68) – now becomes

$$\begin{aligned}
c_1\phi(|a - t_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|a - t_j|) + c_n\phi(|a - t_n|) &= \alpha, \\
c_1 L\phi(|t - t_1|) + \sum_{j=2}^{n-1} c_j LL^{(2)}\phi(|t - t_j|) + c_n L\phi(|t - t_n|) &= f(t), \qquad t \in [a, b], \\
c_1\phi(|b - t_1|) + \sum_{j=2}^{n-1} c_j L^{(2)}\phi(|b - t_j|) + c_n\phi(|b - t_n|) &= \beta.
\end{aligned}$$

If we enforce the collocation conditions at the interior points $t_2, \ldots, t_{n-1}$, then we get the system of linear equations

$$\begin{bmatrix}
\phi(|a - t_1|) & L^{(2)}\phi(|a - t_2|) & \ldots & L^{(2)}\phi(|a - t_{n-1}|) & \phi(|a - t_n|) \\
L\phi(|t_2 - t_1|) & LL^{(2)}\phi(|t_2 - t_2|) & \ldots & LL^{(2)}\phi(|t_2 - t_{n-1}|) & L\phi(|t_2 - t_n|) \\
\vdots & & & \vdots & \\
L\phi(|t_{n-1} - t_1|) & LL^{(2)}\phi(|t_{n-1} - t_2|) & \ldots & LL^{(2)}\phi(|t_{n-1} - t_{n-1}|) & L\phi(|t_{n-1} - t_n|) \\
\phi(|b - t_1|) & L^{(2)}\phi(|b - t_2|) & \ldots & L^{(2)}\phi(|b - t_{n-1}|) & \phi(|b - t_n|)
\end{bmatrix} \times$$

$$\times \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} \alpha \\ f(t_2) \\ \vdots \\ f(t_{n-1}) \\ \beta \end{bmatrix}. \tag{69}$$

Remark 1. The matrix in (69) is symmetric as claimed earlier. This is obvious if L is a differential operator of even order. For odd-order terms one can see that while differentiation with respect to the second variable introduces a sign change, this sign change is canceled by an interchange of the arguments so that $L\phi(|t_i - t_j|) = L^{(2)}\phi(|t_j - t_i|)$ (see the example below).

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

$$Ly(t) = y''(t) + wy'(t) + vy(t),$$

and multiquadrics. Then

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

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 $w = v = 0$ 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 Other basis functions such as polynomials or splines are also frequently used for collocation. In particular, the use of polynomials leads to so-called *spectral* or *pseudo-spectral* methods.

10 Pseudospectral Methods for Two-Point BVPs

Another class of very accurate numerical methods for BVPs (as well as many time-dependent PDEs) are the so-called *spectral* or *pseudospectral methods*. The basic idea is similar to the collocation method described above. However, now we use other basis functions. The following discussion closely follows the first few chapters of Nick Trefethen's book "Numerical Methods in Matlab".

Before we go into any details we present an example.

Example Consider the simple linear 2-pt BVP

$$y''(t) = e^{4t}, \quad t \in (-1, 1)$$

with boundary conditions $y(-1) = y(1) = 0$. The analytic solution of this problem is given by

$$y(t) = \left[e^{4t} - t\sinh(4) - \cosh(4)\right]/16.$$

In the Matlab program `PSBVPDemo.m` we compare the new pseudospectral approach with the finite difference approach.

The high accuracy of the pseudospectral method is impressive, and we use this as our motivation to take a closer look at this method.

As with all the other numerical methods, we require some sort of *discretization*. For pseudospectral methods we do the same as for finite difference methods and the RBF collocation methods, i.e., we introduce a set of grid points $t_1, t_2, \ldots, t_N$ in the interval of interest.

10.1 Differentiation Matrices

The main ingredient for pseudospectral methods is the concept of a *differentiation matrix* D. This matrix will map a vector of function values $\boldsymbol{y} = [y(t_1), \ldots, y(t_N)]^T = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N]^T$ at the grid points to a vector $\boldsymbol{y}'$ of derivative values, i.e.,

$$\boldsymbol{y}' = D\boldsymbol{y}.$$

What does such a differentiation matrix look like? Let's assume that the grid points are uniformly spaced with spacing $t_{j+1} - t_j = h$ for all j, and that the vector of function values $\boldsymbol{y}$ comes from a periodic function so that we can add the two auxiliary values $\boldsymbol{y}_0 = \boldsymbol{y}_N$ and $\boldsymbol{y}_{N+1} = \boldsymbol{y}_1$.

In order to approximate the derivative $y'(t_j)$ we start with another look at the finite difference approach. We use the symmetric (second-order) finite difference approximation

$$y'(t_j) \approx \boldsymbol{y}'_j = \frac{\boldsymbol{y}_{j+1} - \boldsymbol{y}_{j-1}}{2h}, \qquad j = 1, \ldots, N.$$

Note that this formula also holds at both ends ($j = 1$ and $j = N$) since we are assuming periodicity of the data.

These equations can be collected in matrix-vector form:

$$\boldsymbol{y}' = D\boldsymbol{y}$$

with $\boldsymbol{y}$ and $\boldsymbol{y}'$ as above and

$$D = \frac{1}{h}\begin{bmatrix} 0 & \frac{1}{2} & & & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \ddots & & \\ & & \ddots & & \\ & & \ddots & 0 & \frac{1}{2} \\ \frac{1}{2} & & & -\frac{1}{2} & 0 \end{bmatrix}.$$

Remark This matrix has a very special structure. It is both *Toeplitz* and *circulant*. In a Toeplitz matrix the entries in each diagonal are constant, while a circulant matrix is generated by a single row vector whose entries are shifted by one (in a circulant manner) each time a new row is generated. As we will see later, the fast Fourier transform (FFT) can deal with such matrices in a particularly efficient manner.

As we saw earlier, there is a close connection between finite difference approximations of derivatives and polynomial interpolation. For example, the symmetric 2nd-order approximation used above can also be obtained by differentiating the interpolating polynomial p of degree 2 to the data $\{(t_{j-1}, \boldsymbol{y}_{j-1}), (t_j, \boldsymbol{y}_j), (t_{j+1}, \boldsymbol{y}_{j+1})\}$, and then evaluating at $t = t_j$.

We can also use a degree 4 polynomial to interpolate the 5 (symmetric) pieces of data $\{(t_{j-2}, \boldsymbol{y}_{j-2}), (t_{j-1}, \boldsymbol{y}_{j-1}), (t_j, \boldsymbol{y}_j), (t_{j+1}, \boldsymbol{y}_{j+1}), (t_{j+2}, \boldsymbol{y}_{j+2})\}$. This leads to (e.g., modifying the code in the Maple worksheet `472_DerivativeEstimates.mws`)

$$y'(t_j) \approx \boldsymbol{y}'_j = -\frac{\boldsymbol{y}_{j+2} - 8\boldsymbol{y}_{j+1} + 8\boldsymbol{y}_{j-1} - \boldsymbol{y}_{j-2}}{12h}, \qquad j = 1, \ldots, N,$$

so that we get the differentiation matrix

$$D = \frac{1}{h}\begin{bmatrix} 0 & \frac{2}{3} & -\frac{1}{12} & & & \frac{1}{12} & -\frac{2}{3} \\ -\frac{2}{3} & 0 & \frac{2}{3} & -\frac{1}{12} & & & \frac{1}{12} \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ -\frac{1}{12} & & & \frac{1}{12} & -\frac{2}{3} & 0 & \frac{2}{3} \\ \frac{2}{3} & -\frac{1}{12} & & & \frac{1}{12} & -\frac{2}{3} & 0 \end{bmatrix}.$$

Note that this matrix is again a circulant Toeplitz matrix (since the data is assumed to be periodic). However, now there are 5 diagonals, instead of the 3 for the second-order example above.

Example The fourth-order convergence of the finite-difference approximation above is illustrated in the Matlab script `FD4Demo.m`.

It should now be clear that – in order to increase the accuracy of the finite-difference derivative approximation to spectral order – we want to keep on increasing the polynomial degree so that more and more grid points are being used, and the differentiation matrix becomes a dense matrix. Thus, we can think of pseudospectral methods as finite difference methods based on *global* polynomial interpolants instead of local ones.

For an infinite interval with infinitely many grid points spaced a distance h apart one can show that the resulting differentiation matrix is given by the circulant Toeplitz matrix

$$D = \frac{1}{h} \begin{bmatrix} & \vdots & \\ \ddots & \frac{1}{3} & \\ \ddots & -\frac{1}{2} & \\ \ddots & 1 & \\ & 0 & \\ & -1 & \ddots \\ & \frac{1}{2} & \ddots \\ & -\frac{1}{3} & \ddots \\ & \vdots & \end{bmatrix}. \tag{70}$$

For a finite (even) N and periodic data we will show later that the differentiation matrix is given by

$$D_N = \begin{bmatrix} & \vdots & \\ \ddots & \frac{1}{2}\cot\frac{3h}{2} & \\ \ddots & -\frac{1}{2}\cot\frac{2h}{2} & \\ \ddots & \frac{1}{2}\cot\frac{1h}{2} & \\ & 0 & \\ & -\frac{1}{2}\cot\frac{1h}{2} & \ddots \\ & \frac{1}{2}\cot\frac{2h}{2} & \ddots \\ & -\frac{1}{2}\cot\frac{3h}{2} & \ddots \\ & \vdots & \end{bmatrix}. \tag{71}$$

Example If $N = 4$, then we have

$$D_4 = \begin{bmatrix} 0 & \frac{1}{2}\cot\frac{1h}{2} & \frac{1}{2}\cot\frac{2h}{2} & -\frac{1}{2}\cot\frac{1h}{2} \\ -\frac{1}{2}\cot\frac{1h}{2} & 0 & \frac{1}{2}\cot\frac{1h}{2} & \frac{1}{2}\cot\frac{2h}{2} \\ \frac{1}{2}\cot\frac{2h}{2} & -\frac{1}{2}\cot\frac{1h}{2} & 0 & \frac{1}{2}\cot\frac{1h}{2} \\ \frac{1}{2}\cot\frac{1h}{2} & \frac{1}{2}\cot\frac{2h}{2} & -\frac{1}{2}\cot\frac{1h}{2} & 0 \end{bmatrix}.$$

The Matlab script `PSDemo.m` illustrates the spectral convergence obtained with the matrix D_N for various values of N. The output should be compared with that of the previous example `FD4Demo.m`.

10.2 Unbounded Grids and the Semi-Discrete Fourier Transform

We now consider an *infinite* uniform grid $h\mathbb{Z}$ with grid points $t_j = jh$ for all integers j. While this case is not useful for practical computation, it is important for our understanding of problems on bounded intervals.

First we recall the definition of the *Fourier transform* $\hat{y}$ of a function y that is square-integrable on $\mathbb{R}$:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} y(t) dt, \qquad \omega \in \mathbb{R}\,. \tag{72}$$

Conversely, the *inverse Fourier transform* lets us reconstruct y from its Fourier transform $\hat{y}$:

$$y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega t} \hat{y}(\omega) d\omega, \qquad t \in \mathbb{R}\,. \tag{73}$$

Example Consider the function

$$y(t) = \begin{cases} 1, & \text{if } -1/2 \le t \le 1/2 \\ 0, & \text{otherwise,} \end{cases}$$

and compute its Fourier transform.

By the definition of the Fourier transform, the definition of y and Euler's formula we have

$$\begin{aligned} \hat{y}(\omega) &= \int_{-\infty}^{\infty} e^{-i\omega t} y(t) dt \\ &= \int_{-1/2}^{1/2} e^{-i\omega t} dt \\ &= \int_{-1/2}^{1/2} \left[\cos(\omega t) - i \sin(\omega t)\right] dt \\ &= 2 \int_{0}^{1/2} \cos(\omega t) dt \\ &= 2 \frac{\sin(\omega t)}{\omega}\Big|_0^{1/2} = \frac{\sin \omega/2}{\omega/2}. \end{aligned}$$

These functions play an important role in many applications (e.g., signal processing). The function y is known as a *square pulse* or *characteristic function of the interval* $[-1/2, 1/2]$, and its Fourier transform $\hat{y}$ is known as the *sinc function.*

If we restrict our attention to a *discrete* (unbounded) physical space, i.e., the function y is now given by the (infinite) vector $\boldsymbol{y} = [\ldots, \boldsymbol{y}_{-1}, \boldsymbol{y}_0, \boldsymbol{y}_1, \ldots]^T$ of discrete values, then the formulas change. In fact, the *semidiscrete Fourier transform* of $\boldsymbol{y}$ is given by the (continuous) function

$$\hat{y}(\omega) = h \sum_{j=-\infty}^{\infty} e^{-i\omega t_j} \boldsymbol{y}_j, \qquad \omega \in [-\pi/h, \pi/h], \tag{74}$$

and the *inverse semidiscrete Fourier transform* is given by the (discrete infinite) vector $\boldsymbol{y}$ whose components are of the form

$$\boldsymbol{y}_j = \frac{1}{2\pi} \int_{-\pi/h}^{\pi/h} e^{i\omega t_j} \hat{y}(\omega) d\omega, \qquad j \in \mathbb{Z}. \tag{75}$$

Remark Note that the notion of a semidiscrete Fourier transform is just a different name for a *Fourier series* based on the complex exponentials $e^{-i\omega t_j}$ with *Fourier coefficients* $\boldsymbol{y}_j$.

The interesting difference between the continuous and semidiscrete setting is marked by the *bounded* Fourier space in the semidiscrete setting. This can be explained by the phenomenon of *aliasing*. Aliasing arises when a continuous function is sampled on a discrete set. In particular, the two complex exponential functions $f(t) = e^{i\omega_1 t}$ and $g(t) = e^{i\omega_2 t}$ differ from each other on the real line as long as $\omega_1 \neq \omega_2$. However, if we sample the two functions on the grid $h\mathbb{Z}$, then we get the vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ with values $\boldsymbol{f}_j = e^{i\omega_1 t_j}$ and $\boldsymbol{g}_j = e^{i\omega_2 t_j}$. Now, if $\omega_2 = \omega_1 + 2k\pi/h$ for some integer k, then $\boldsymbol{f}_j = \boldsymbol{g}_j$ for all j, and the two (different) continuous functions f and g appear identical in their discrete representations $\boldsymbol{f}$ and $\boldsymbol{g}$. Thus, any complex exponential $e^{i\omega t}$ is matched on the grid $h\mathbb{Z}$ by infinitely many other complex exponentials (its *aliases*). Therefore we can limit the representation of the Fourier variable ω to an interval of length $2\pi/h$. For reasons of symmetry we use $[-\pi/h, \pi/h]$.

10.2.1 Spectral Differentiation

To get the interpolant of the $\boldsymbol{y}_j$ values we can now use an extension of the inverse semidiscrete Fourier transform, i.e., we define the interpolant to be the function

$$p(t) = \frac{1}{2\pi} \int_{-\pi/h}^{\pi/h} e^{i\omega t} \hat{y}(\omega) d\omega, \qquad t \in \mathbb{R}\,. \tag{76}$$

It is obvious from this definition that p interpolates the data, i.e., $p(t_j) = \boldsymbol{y}_j$, for any $j \in \mathbb{Z}$.

Moreover, the Fourier transform of the function p turns out to be

$$\hat{p}(\omega) = \begin{cases} \hat{y}(\omega), & \omega \in [\pi/h, \pi, h] \\ 0, & \text{otherwise} \end{cases}$$

This kind of function is known as a *band-limited function*.

The spectral derivative vector $\boldsymbol{y}'$ of $\boldsymbol{y}$ can now be obtained by one of the following two procedures we are about to present. First,

1. Sample the function y at the (infinite set of) discrete points $t_j \in h\mathbb{Z}$ to obtain the data vector $\boldsymbol{y}$ with components $\boldsymbol{y}_j$.

2. Compute the semidiscrete Fourier transform of the data via (74):

$$\hat{y}(\omega) = h \sum_{j=-\infty}^{\infty} e^{-i\omega t_j} \boldsymbol{y}_j, \qquad \omega \in [-\pi/h, \pi/h].$$

3. Find the band-limited interpolant p of the data $\boldsymbol{y}_j$ via (76).

4. Differentiate p and evaluate at the t_j.

However, from a computational point of view it is better to deal with this problem in the Fourier domain. We begin by noting that the Fourier transform of the derivative y' is given by

$$\widehat{y'}(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} y'(t) dt.$$

Applying integration by parts we get

$$\widehat{y'}(\omega) = e^{-i\omega t} y(t)\big|_{-\infty}^{\infty} + i\omega \int_{-\infty}^{\infty} e^{-i\omega t} y(t) dt.$$

If $y(t)$ tends to zero for $t \to \pm\infty$ (which it has to for the Fourier transform to exist) then we see that

$$\widehat{y'}(\omega) = i\omega \hat{y}(\omega). \tag{77}$$

Therefore, we obtain the spectral derivative $\boldsymbol{y'}$ by the following alternate procedure:

1. Sample the function y at the (infinite set of) discrete points $t_j \in h\mathbb{Z}$ to obtain the data vector $\boldsymbol{y}$ with components $\boldsymbol{y}_j$.

2. Compute the semidiscrete Fourier transform of the data via (74):

$$\hat{y}(\omega) = h \sum_{j=-\infty}^{\infty} e^{-i\omega t_j} \boldsymbol{y}_j, \qquad \omega \in [-\pi/h, \pi/h].$$

3. Compute the Fourier transform of the derivative via (77):

$$\widehat{y'}(\omega) = i\omega \hat{y}(\omega).$$

4. Find the derivative vector via inverse semidiscrete Fourier transform (see (75)), i.e.,

$$\boldsymbol{y}'_j = \frac{1}{2\pi} \int_{-\pi/h}^{\pi/h} e^{i\omega t_j} \widehat{y'}(\omega) d\omega, \qquad j \in \mathbb{Z}.$$

Now we need to find out how we can obtain the entries of the differentiation matrix D from the preceding discussion. We follow the first procedure above.

In order to be able to compute the semidiscrete Fourier transform of an arbitrary data vector $\boldsymbol{y}$ we represent its components in terms of *shifts of* (discrete) *delta functions*, i.e.,

$$\boldsymbol{y}_j = \sum_{k=-\infty}^{\infty} \boldsymbol{y}_k \delta_{j-k}, \tag{78}$$

where the *Kronecker delta function* is defined by

$$\delta_j = \begin{cases} 1 & j = 0 \\ 0 & \text{otherwise.} \end{cases}$$

We use this approach since the semidiscrete Fourier transform of the delta function can be computed easily. In fact, according to (74)

$$\begin{aligned}\hat{\delta}(\omega) &= h\sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\delta_j \\ &= he^{-i\omega t_0} = h\end{aligned}$$

for all $\omega \in [-\pi/h, \pi/h]$. Then the band-limited interpolant of δ is of the form (see (76))

$$\begin{aligned}p(t) &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}\hat{\delta}(\omega)d\omega \\ &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}hd\omega \\ &= \frac{h}{\pi}\int_{0}^{\pi/h} \cos(\omega t)d\omega \\ &= \frac{h}{\pi}\frac{\sin(\omega t)}{t}\Big|_0^{\pi/h} \\ &= \frac{h}{\pi}\frac{\sin(\pi t/h)}{t} = \frac{\sin(\pi t/h)}{\pi t/h} = \mathrm{sinc}(\pi t/h).\end{aligned}$$

Therefore, the band-limited interpolant of an arbitrary data vector $\boldsymbol{y}$ is given by

$$\begin{aligned}p(t) &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}\hat{y}(\omega)d\omega \\ &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}\left[h\sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\boldsymbol{y}_j\right]d\omega \\ &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}\left[h\sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\sum_{k=-\infty}^{\infty}\boldsymbol{y}_k\delta_{j-k}\right]d\omega.\end{aligned}$$

Thus far we have used the definition of the band-limited interpolant (76), the definition of the semidiscrete Fourier transform of y (74), and the representation (78). Interchanging the summation, and then using the definition of the delta function and the same calculation as for the band-limited interpolant of the delta function above we obtain the final form of the band-limited interpolant of an arbitrary data vector $\boldsymbol{y}$ as

$$\begin{aligned}p(t) &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}\left[h\sum_{k=-\infty}^{\infty}\boldsymbol{y}_k\sum_{j=-\infty}^{\infty} e^{-i\omega t_j}\delta_{j-k}\right]d\omega \\ &= \frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega t}h\sum_{k=-\infty}^{\infty}\boldsymbol{y}_k e^{-i\omega t_k}d\omega \\ &= \sum_{k=-\infty}^{\infty}\boldsymbol{y}_k\frac{1}{2\pi}\int_{-\pi/h}^{\pi/h} e^{i\omega(t-t_k)}hd\omega \\ &= \sum_{k=-\infty}^{\infty}\boldsymbol{y}_k\mathrm{sinc}\frac{(t-t_k)\pi}{h}.\end{aligned}$$

Example Band-limited interpolation for the functions

$$y_1(t) = \begin{cases} 1, & t = 0 \\ 0, & \text{otherwise,} \end{cases}$$

$$y_2(t) = \begin{cases} 1, & |t| \le 3 \\ 0, & \text{otherwise,} \end{cases}$$

and

$$y_3(t) = (1 - |t|/3)_+.$$

is illustrated in the Matlab script `BandLimitedDemo.m`. Note that the accuracy of the reproduction is not very high. Note, in particular, the *Gibbs phenomenon* that arises for $h \to 0$. This is due to the low smoothness of the data functions.

In order to get the components of the derivative vector $\boldsymbol{y}'$ we need to differentiate the band-limited interpolant and evaluate at the grid points. By linearity this leads to

$$\boldsymbol{y}_j = p'(t_j) = \sum_{k=-\infty}^{\infty} \boldsymbol{y}_k \frac{d}{dt} \left[\operatorname{sinc} \frac{(t - t_k)\pi}{h} \right]_{t=t_j},$$

or in (infinite) matrix form

$$\boldsymbol{y}' = D\boldsymbol{y}$$

with the entries of D given by

$$D_{jk} = \frac{d}{dt} \left[\operatorname{sinc} \frac{(t - t_k)\pi}{h} \right]_{t=t_j}, \qquad j, k = -\infty, \ldots, \infty.$$

The entries in the $k = 0$ column of D are of the form

$$D_{j0} = \frac{d}{dt} \left[\operatorname{sinc} \frac{t\pi}{h} \right]_{t=t_j=jh} = \begin{cases} 0, & j = 0 \\ \frac{(-1)^j}{jh}, & \text{otherwise,} \end{cases}$$

The remaining columns are shifts of this column since the matrix is a Toeplitz matrix. This is exactly of the form (70). The explicit formula for the derivative of the sinc function above is obtained using elementary calculations:

$$\frac{d}{dt} \left[\operatorname{sinc} \frac{t\pi}{h} \right] = \frac{1}{t} \cos\left(\frac{t\pi}{h} \right) - \frac{h}{t^2\pi} \sin\left(\frac{t\pi}{h} \right),$$

so that

$$\frac{d}{dt} \left[\operatorname{sinc} \frac{t\pi}{h} \right]_{t=t_j=jh} = \frac{1}{jh} \cos(j\pi) - \frac{1}{j^2 h\pi} \sin(j\pi).$$

10.3 Periodic Grids: The DFT and FFT

We now consider the case of a *bounded* grid with periodic data, i.e., we will now explain how to find the entries in the matrix D_N of (71).

To keep the discussion simple we will consider the interval $[0, 2\pi]$ only, and assume that we are given N (with N even) uniformly spaced grid points $t_j = jh$, $j = 1, \ldots, N$, with $h = 2\pi/N$.

Remark Formulas for odd N also exist, but are slightly different. For the sake of clarity, we focus only on the even case here.

As in the previous subsection we now look at the Fourier transform of the discrete and periodic data $\boldsymbol{y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N]^T$ with $\boldsymbol{y}_j = y(jh) = y(2j\pi/N)$, $j = 1, \ldots, N$. For the same reason of aliasing the Fourier domain will again be bounded. Moreover, the periodicity of the data implies that the Fourier domain is also discrete (since only waves e^{ikt} with integer wavenumber k have period 2π).

Thus, the *discrete Fourier transform* (DFT) is given by

$$\hat{\boldsymbol{y}}_k = h \sum_{j=1}^{N} e^{-ikt_j} \boldsymbol{y}_j, \qquad k = -\frac{N}{2} + 1, \ldots, \frac{N}{2}. \tag{79}$$

Note that the (continuous) Fourier domain $[\pi/h, \pi/h]$ used earlier now translates to the discrete domain noted in (79) since $h = 2\pi/N$ is equivalent to $\pi/h = N/2$.

The formula for the *inverse discrete Fourier transform* (inverse DFT) is given by

$$\boldsymbol{y}_j = \frac{1}{2\pi} \sum_{k=-N/2+1}^{N/2} e^{ikt_j} \hat{\boldsymbol{y}}_k, \qquad j = 1, \ldots, N. \tag{80}$$

We obtain the spectral derivative of the finite vector data by exactly the same procedure as in the previous subsection. First, we need the band-limited interpolant of the data. It is given by the formula

$$p(t) = \frac{1}{2\pi} \sum_{k=-N/2}^{N/2}{}' e^{ikt} \hat{\boldsymbol{y}}_k, \qquad t \in [0, 2\pi]. \tag{81}$$

Here we *define* $\hat{\boldsymbol{y}}_{-N/2} = \hat{\boldsymbol{y}}_{N/2}$, and the prime on the sum indicates that we add the first and last summands only with weight $1/2$. This modification is required for the band-limited interpolant to work properly.

Remark The band-limited interpolant is actually a *trigonometric polynomial* of degree $N/2$, i.e., $p(t)$ can be written as a linear combination of the trigonometric functions $1, \sin t, \cos t, \sin 2t, \cos 2t, \ldots, \sin Nt/2, \cos Nt/2$. We will come back to this fact when we discuss non-periodic data.

Next, we want to represent an arbitrary periodic data vector $\boldsymbol{y}$ as a linear combination of shifts of periodic delta functions. We omit the details here (they can be found

in the Trefethen book) and give only the formula for the band-limited interpolant of the periodic delta function:

$$p(t) = S_N(t) = \frac{\sin(\pi t/h)}{(2\pi/h)\tan(t/2)},$$

which is known as the *periodic sinc function* S_N.

Now, just as in the previous subsection, the band-limited interpolant for an arbitrary data function can be written as

$$p(t) = \sum_{k=1}^{N} \boldsymbol{y}_k S_N(t - t_k).$$

Finally, using the same arguments and similar elementary calculations as earlier, we get

$$S'_N(t_j) = \begin{cases} 0, & j \equiv 0 \pmod N, \\ \frac{1}{2}(-1)^j \cot(jh/2), & j \not\equiv 0 \pmod N. \end{cases}$$

These are the entries of the N-th column of the Toeplitz matrix (71).

Example The Matlab script `SpectralDiffDemo.m` illustrates the use of spectral differentiation for the not so smooth hat function and for the infinitely smooth function $y(t) = e^{\sin t}$.

10.3.1 Implementation via FFT

The most efficient computational approach is to view spectral differentiation in the Fourier domain (the alternate approach earlier) and then implement the DFT via the fast Fourier transform (FFT). The general outline is as follows:

1. Sample the function y at the (finite set of) discrete points t_j, $j = 1, \ldots, N$ to obtain the data vector $\boldsymbol{y}$ with components $\boldsymbol{y}_j$.

2. Compute the discrete Fourier transform of the (finite) data vector via (79):

$$\hat{\boldsymbol{y}}_k = h \sum_{j=1}^{N} e^{-ikt_j} \boldsymbol{y}_j, \qquad k = -\frac{N}{2} + 1, \ldots, \frac{N}{2}.$$

3. Compute the Fourier transform of the derivative based on (77), i.e.,

$$\widehat{\boldsymbol{y}'}_k = \begin{cases} 0, & k = N/2, \\ ik\hat{\boldsymbol{y}}_k, & \text{otherwise.} \end{cases}$$

4. Find the derivative vector via inverse discrete Fourier transform (see (80)), i.e.,

$$\boldsymbol{y}'_j = \frac{1}{2\pi} \sum_{k=-N/2+1}^{N/2} e^{ikt_j} \widehat{\boldsymbol{y}'}_k, \qquad j = 1, \ldots, N.$$

Remark Cooley and Tukey (1965) are usually given credit for discovering the FFT. However, the same algorithm was already known to Gauss (even before Fourier completed his work on what is known today as the Fourier transform). A detailed discussion of this algorithm goes beyond the scope of this course. We simply use the Matlab implementations `fft` and `ifft`. These implementations are based on the current state-of-the-art FFTW algorithm (the "fastest Fourier transform in the West") developed at MIT by Matteo Frigo and Steven G. Johnson.

Example The Matlab script `SpectralDiffFFTDemo.m` is an FFT version of the earlier script `SpectralDiffDemo.m`. The FFT implementation is considerably faster than the implementation based on differentiation matrices (see Computer Assignment 5).

10.4 Smoothness and Spectral Accuracy

Without getting into any details (see Chapter 4 of Trefethen's book) we will simply illustrate with a few examples the basic behavior of spectral differentiation:

> The smoother the data, the more accurate the spectral derivative.

Example In the Matlab script `SpectralAccuracyDemo.m` we expand on the earlier script `SpectralDiffDemo.m` and illustrate the dependence of the convergence rate of spectral differentiation on the smoothness of the data more clearly for the four periodic functions on $[0, 2\pi]$

$$\begin{aligned} y_1(t) &= |\sin t|^3, \\ y_2(t) &= \exp(-\sin^{-2}(t/2)), \\ y_3(t) &= \frac{1}{1+\sin^2(t/2)}, \\ y_4(t) &= \sin(10t). \end{aligned}$$

These functions are arranged according to their (increasing) smoothness. The function y_1 has a third derivative of bounded variation, y_2 is infinitely differentiable (but not analytic), y_3 is analytic in the strip $|\mathrm{Im}(t)| < 2\ln(1+\sqrt{2})$ in the complex plane, and y_4 is band-limited.
Note: A continuous function y is of bounded variation if

$$\sup_{t_0<t_1<\cdots<t_N} \sum_{j=1}^{N} |y(t_j) - y(t_{j-1})|$$

is bounded for all choices of $t_0, t_1, \ldots, t_N$. Plainly said, a function of bounded variation cannot "wiggle around too much". For example, on the interval $[0, 1/2]$ the function $y(t) = t^2 \sin(1/t)$ is of bounded variation while $y(t) = t\sin(1/t)$ is not.

10.5 Polynomial Interpolation and Clustered Grids

We already saw in the Matlab script `BandLimitedDemo.m` that a spectral interpolant performs very poorly for non-smooth functions. Thus, if we just went ahead and treated a problem on a bounded domain as a periodic problem via periodic extension, then the

resulting jumps that may arise at the endpoints of the original interval would lead to Gibbs phenomena and a significant degradation of accuracy. Therefore, we do not use the trigonometric polynomials (discrete Fourier transforms) but algebraic polynomials instead.

For interpolation with algebraic polynomials we saw at the very beginning of this course (in the Matlab script `PolynomialInterpolationDemo.m`) the effect that different distributions of the interpolation nodes in a bounded interval have on the accuracy of the interpolant (the so-called *Runge phenomenon*). Clearly, the accuracy is much improved if the points are clustered near the endpoints of the interval. In fact, the so-called *Chebyshev points*

$$t_j = \cos(j\pi/N), \qquad j = 0, 1, \ldots, N$$

yield a set of such clustered interpolation nodes on the standard interval $[-1, 1]$. These points can easily be mapped by a linear transformation to any other interval $[a, b]$ (see Assignment 8). Chebyshev points arise often in numerical analysis. They are the extremal points of the so-called *Chebyshev polynomials* (a certain type of *orthogonal polynomial*). In fact, Chebyshev points are *equally spaced on the unit circle*, and therefore one can observe a nice connection between spectral differentiation on bounded intervals with Chebyshev points and periodic problems on bounded intervals as described earlier. It turns out that (contrary to our expectations) the FFT can also be used for the Chebyshev case. However, we will only consider Chebyshev differentiation matrices below.

10.6 Chebyshev Differentiation Matrices

Our last step in our preparation for the solution of general boundary value problems is to determine the entries of the differentiation matrices to be used for problems on bounded intervals (with non-periodic data).

As before, we follow our well-established approach for spectral differentiation:

1. Discretize the interval $[-1, 1]$ using the Chebyshev points

   $$t_j = \cos(j\pi/N), \qquad j = 0, 1, \ldots, N,$$

   and sample the function y at those points to obtain the data vector $\boldsymbol{y} = [y(t_0), y(t_1), \ldots, y(t_N)]^T$.

2. Find the (algebraic) polynomial p of degree at most N that interpolates the data, i.e., s.t.

   $$p(t_i) = \boldsymbol{y}_i, \qquad i = 0, 1, \ldots, N.$$

3. Obtain the spectral derivative vector $\boldsymbol{y}'$ by differentiating p and evaluating at the grid points:

   $$\boldsymbol{y}'_i = p'(t_i), \qquad i = 0, 1, \ldots, N.$$

This procedure (implicitly) defines the differentiation matrix D_N that gives us

$$\boldsymbol{y}' = D_N \boldsymbol{y}.$$

Before we look at the general formula for the entries of D_N we consider some simple examples.

Example For $N = 1$ we have the two points $t_0 = 1$ and $t_1 = -1$, and the interpolant is given by

$$\begin{aligned} p(t) &= \frac{t - t_1}{t_0 - t_1}\boldsymbol{y}_0 + \frac{t_0 - t}{t_0 - t_1}\boldsymbol{y}_1 \\ &= \frac{t+1}{2}\boldsymbol{y}_0 + \frac{1-t}{2}\boldsymbol{y}_1. \end{aligned}$$

The derivative of p is (the constant)

$$p'(t) = \frac{1}{2}\boldsymbol{y}_0 - \frac{1}{2}\boldsymbol{y}_1,$$

so that we have

$$\boldsymbol{y}' = \begin{bmatrix} \frac{1}{2}\boldsymbol{y}_0 - \frac{1}{2}\boldsymbol{y}_1 \\ \frac{1}{2}\boldsymbol{y}_0 - \frac{1}{2}\boldsymbol{y}_1 \end{bmatrix}$$

and the differentiation matrix is given by

$$D_1 = \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \end{bmatrix}.$$

Example For $N = 2$ we start with the three Chebyshev points $t_0 = 1$, $t_1 = 0$, and $t_2 = -1$. The quadratice interpolating polynomial (in Lagrange form) is given by

$$\begin{aligned} p(t) &= \frac{(t-t_1)(t-t_2)}{(t_0-t_1)(t_0-t_2)}\boldsymbol{y}_0 + \frac{(t-t_0)(t-t_2)}{(t_1-t_0)(t_1-t_2)}\boldsymbol{y}_1 + \frac{(t-t_0)(t-t_1)}{(t_2-t_0)(t_2-t_1)}\boldsymbol{y}_2 \\ &= \frac{t(t+1)}{2}\boldsymbol{y}_0 - (t-1)(t+1)\boldsymbol{y}_1 + \frac{(t-1)t}{2}\boldsymbol{y}_2. \end{aligned}$$

Now the derivative of p is a linear polynomial

$$p'(t) = \left(t + \frac{1}{2}\right)\boldsymbol{y}_0 - 2t\boldsymbol{y}_1 + \left(t - \frac{1}{2}\right)\boldsymbol{y}_2,$$

so that – evaluating at the nodes – we have

$$\boldsymbol{y}' = \begin{bmatrix} \frac{3}{2}\boldsymbol{y}_0 - 2\boldsymbol{y}_1 + \frac{1}{2}\boldsymbol{y}_2 \\ \frac{1}{2}\boldsymbol{y}_0 - \frac{1}{2}\boldsymbol{y}_2 \\ -\frac{1}{2}\boldsymbol{y}_0 + 2\boldsymbol{y}_1 - \frac{3}{2}\boldsymbol{y}_2 \end{bmatrix}$$

and the differentiation matrix is given by

$$D_2 = \begin{bmatrix} \frac{3}{2} & -2 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{2} & 2 & -\frac{3}{2} \end{bmatrix}.$$

We note that the differentiation matrices no longer are Toeplitz or circulant. Instead, the entries satisfy (also in the general case below)

$$(D_N)_{ij} = -(D_N)_{N-i,N-j}.$$

For general N one can prove

Theorem 10.1 *For each $N \geq 1$, let the rows and columns of the $(N+1) \times (N+1)$* Chebyshev spectral differentiation matrix D_N *be indexed from 0 to N. The entries of this matrix are*

$$(D_N)_{00} = \frac{2N^2+1}{6}, \qquad (D_N)_{NN} = -\frac{2N^2+1}{6},$$
$$(D_N)_{jj} = \frac{-t_j}{2(1-t_j^2)}, \qquad j = 1, \ldots, N-1,$$
$$(D_N)_{ij} = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(t_i - t_j)}, \qquad i \neq j, \quad i, j = 0, 1, \ldots, N,$$

where

$$c_i = \begin{cases} 2, & i = 0 \text{ or } N, \\ 1, & \text{otherwise.} \end{cases}$$

This matrix is implemented in the Matlab script `cheb.m` that was already used in the Matlab function `PSBVP.m` that we used in our motivational example `PSBVPDemo.m` at the beginning of this chapter. Note that only the off-diagonal entries are computed via the formulas given in the theorem. For the diagonal entries the formula

$$(D_N)_{ii} = -\sum_{\substack{j=0 \\ j \neq i}}^{N} (D_N)_{ij}$$

was used.

Example The spectral accuracy of Chebyshev differentiation matrices is illustrated in the Matlab script `ChebyshevAccuracyDemo.m`. One should compare this to the earlier script `SpectralAccuracyDemo.m` in the periodic case.

The functions used for the Chebyshev example are

$$\begin{aligned} y_1(t) &= |t|^3, \\ y_2(t) &= \exp(-t^{-2}), \\ y_3(t) &= \frac{1}{1+t^2}, \\ y_4(t) &= t^{10}. \end{aligned}$$

These functions are again arranged according to their (increasing) smoothness. The function y_1 has a third derivative of bounded variation, y_2 is infinitely differentiable (but not analytic), y_3 is analytic in $[-1, 1]$, and y_4 is a polynomial (which corresponds to the band-limited case earlier).

Note that the error for the derivative of the function y_2 dips to zero for $N = 2$ since the true derivative is given by

$$y_2'(t) = 2\frac{\exp(-t^{-2})}{t^3},$$

and the values at $t_0 = 1$, $t_1 = 0$, and $t_2 = -1$ are $2/e$, 0, and $-2/e$, respectively. These all lie on a line (the linear derivative of the quadratic interpolating polynomial).

10.7 Boundary Value Problems

We can now return to our introductory example, the 2-pt boundary value problem

$$y''(t) = e^{4t}, \quad t \in (-1, 1)$$

with boundary conditions $y(-1) = y(1) = 0$. Its analytic solution was given earlier as

$$y(t) = \left[e^{4t} - t\sinh(4) - \cosh(4)\right]/16.$$

How do we solve this problem in the Matlab programs `PSBVPDemo.m` and `PSBVP.m`?

First, we note that – for Chebyshev differentiation matrices – we can obtain higher derivatives by repeated application of the matrix D_N, i.e., if

$$\boldsymbol{y}' = D_N \boldsymbol{y},$$

then

$$\boldsymbol{y}'' = D_N \boldsymbol{y}' = D_N^2 \boldsymbol{y}.$$

In other words, for Chebyshev differentiation matrices

$$D_N^{(k)} = D_N^k, \quad k = 1, \ldots, N,$$

and $D_N^{N+1} = 0$.

Remark We point out that this fact is true only for the Chebyshev case. For the Fourier differentiation matrices we established in the periodic case we in general have $D_N^k \neq D_N^{(k)}$ (see Assignment 8).

With the insight about higher-order Chebyshev differentiation matrices we can view the differential equation above as

$$D_N^2 \boldsymbol{y} = \boldsymbol{f},$$

where the right-hand side vector $\boldsymbol{f} = \exp(4\boldsymbol{t})$, with $\boldsymbol{t} = [t_0, t_1, \ldots, t_N]^T$ the vector of Chebyshev points. This *linear system*, however, cannot be solved uniquely (one can show that the matrix $(N+1) \times (N+1)$ matrix D_N^2 has an $(N+1)$-fold eigenvalue of zero). Of course, this is not a problem. In fact, it is reassuring, since we have not yet taken into account the boundary conditions, and the ordinary *differential equation* (without appropriate boundary conditions) also does not have a unique solution.

So the final question is, how do we deal with the boundary conditions?

We could follow either of two approaches. First, we can build the boundary conditions into the spectral interpolant, i.e.,

1. Take the *interior* Chebyshev points $t_1, \ldots, t_{N-1}$ and form the polynomial interpolant of degree at most N that satisfies the boundary conditions $p(-1) = p(1) = 0$ and interpolates the data vector at the interior points, i.e., $p(t_j) = \boldsymbol{y}_j$, $j = 1, \ldots, N-1$.

2. Obtain the spectral derivative by differentiating p and evaluating at the interior points, i.e.,
$$\boldsymbol{y}_j'' = p''(t_j), \quad j = 1, \ldots, N-1.$$

3. Identify the $(N-1)\times(N-1)$ matrix $\widetilde{D}_N^2$ from the previous relation, and solve the linear system

$$\widetilde{D}_N^2 \boldsymbol{y}(1:N-1) = \exp(4\boldsymbol{t}(1:N-1)),$$

where we used Matlab-like notation.

The second approach is much simpler to implement, but not as straightforward to understand/derive. Since we already know the value of the solution at the boundary, i.e., $\boldsymbol{y}_0 = 0$ and $\boldsymbol{y}_N = 0$, we do not need to include these values in our computation. Moreover, the values of the derivative at the endpoints are of no interest to us. Therefore, we can simply solve the linear system

$$\widetilde{D}_N^2 \boldsymbol{y}(1:N-1) = \exp(4\boldsymbol{t}(1:N-1)),$$

where

$$\widetilde{D}_N^2 = D_N^2(1:N-1, 1:N-1).$$

This is exactly what was done in the Matlab program `PSBVP.m`.

Remark One can show that the eigenvalues of $\widetilde{D}_N^2$ are given by $\lambda_n = -\frac{\pi^2 n^2}{4}$, $n = 1, 2, \ldots, N-1$. Clearly, these values are all nonzero, and the problem has (as it should have) a unique solution.

We are now ready to deal with more complicated boundary value problems. They can be nonlinear, have non-homogeneous boundary conditions, or mixed-type boundary conditions with derivative values specified at the boundary. We give examples for each of these cases.

Example As for our initial value problems earlier, a nonlinear ODE-BVP will be solved by iteration (either fixed-point, or Newton).

Consider

$$y''(t) = e^{y(t)}, \quad t \in (-1, 1)$$

with boundary conditions $y(-1) = y(1) = 0$. In the Matlab program `NonlinearPSBVPDemo.m` we use fixed-point iteration to solve this problem.

Example Next, we consider a linear BVP with non-homogeneous boundary conditions:

$$y''(t) = e^{4t}, \quad t \in (-1, 1)$$

with boundary conditions $y(-1) = 0$, $y(1) = 1$. In the Matlab program `PSBVPNonHomoBCDemo.m` this is simply done by replacing the first and last rows of the differentiation matrix `D2` by corresponding rows of the identity matrix and then imposing the boundary values in the first and last entries of the right-hand side vector `f`.

Example For a linear BVP with mixed boundary conditions such as

$$y''(t) = e^{4t}, \quad t \in (-1, 1)$$

with boundary conditions $y'(-1) = y(1) = 0$ we can follow the same strategy as in the previous example. Now, however, we need to replace the row of `D2` that corresponds to the derivative boundary condition with a row from the first-order differentiation matrix `D`. This leads to the Matlab program `PSBVPMixedBCDemo.m`.

11 Galerkin and Ritz Methods for Elliptic PDEs

11.1 Galerkin Method

We begin by introducing a generalization of the collocation method we saw earlier for two-point boundary value problems. Consider the elliptic PDE

$$Lu(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{82}$$

where L is a linear elliptic partial differential operator such as the Laplacian

$$L = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}, \quad \boldsymbol{x} = (x, y, z) \in \mathbb{R}^3 .$$

At this point we will not worry about the boundary conditions that should be posed with (82).

As with the collocation method discussed earlier, we will obtain the approximate solution in the form of a function (instead of as a collection of discrete values). Therefore, we need an approximation space $\mathcal{U} = \text{span}\{u_1, \ldots, u_n\}$, so that we are able to represent the approximate solution as

$$u = \sum_{j=1}^{n} c_j u_j, \quad u_j \in \mathcal{U}. \tag{83}$$

Using the linearity of L we have

$$Lu = \sum_{j=1}^{n} c_j L u_j.$$

We now need to come up with n (linearly independent) conditions to determine the n unknown coefficients c_j in (83). If $\{\Phi_1, \ldots, \Phi_n\}$ is a linearly independent set of linear functionals, then

$$\Phi_i \left[\sum_{j=1}^{n} c_j L u_j - f \right] = 0, \quad i = 1, \ldots, n, \tag{84}$$

is an appropriate set of conditions. In fact, this leads to a system of linear equations

$$Ac = b$$

with matrix

$$A = \begin{bmatrix} \Phi_1 L u_1 & \Phi_1 L u_2 & \ldots & \Phi_1 L u_n \\ \Phi_2 L u_1 & \Phi_2 L u_2 & \ldots & \Phi_2 L u_n \\ \vdots & \vdots & & \vdots \\ \Phi_n L u_1 & \Phi_n L u_2 & \ldots & \Phi_n L u_n \end{bmatrix},$$

coefficient vector $c = [c_1, \ldots, c_n]^T$, and right-hand side vector

$$b = \begin{bmatrix} \Phi_1 f \\ \Phi_2 f \\ \vdots \\ \Phi_n f \end{bmatrix}.$$

Two popular choices are

1. *Point evaluation functionals*, i.e., $\Phi_i(u) = u(\boldsymbol{x}_i)$, where $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is a set of points chosen such that the resulting conditions are linearly independent, and u is some function with appropriate smoothness. With this choice (84) becomes

$$\sum_{j=1}^{n} c_j L u_j(\boldsymbol{x}_i) = f(\boldsymbol{x}_i), \quad i = 1, \ldots, n,$$

and we now have an extension of the *collocation method* discussed in Chapter 9 to elliptic PDEs is the multi-dimensional setting.

2. If we let $\Phi_i(u) = \langle u, v_i \rangle$, an inner product of the function u with an appropriate *test function* v_i, then (84) becomes

$$\sum_{j=1}^{n} c_j \langle L u_j, v_i \rangle = \langle f, v_i \rangle, \quad i = 1, \ldots, n.$$

If $v_i \in \mathcal{U}$ then this is the classical *Galerkin method*, otherwise it is known as the *Petrov-Galerkin method*.

11.2 Ritz-Galerkin Method

For the following discussion we pick as a model problem a multi-dimensional Poisson equation with homogeneous boundary conditions, i.e.,

$$\begin{aligned} -\nabla^2 u &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{85}$$

with domain $\Omega \subset \mathbb{R}^d$. This problem describes, e.g., the steady-state solution of a vibrating membrane (in the case $d = 2$ with shape Ω) fixed at the boundary, and subjected to a vertical force f.

The first step for the Ritz-Galerkin method is to obtain the *weak form* of (85). This is accomplished by choosing a function v from a space $\mathcal{U}$ of smooth functions, and then forming the inner product of both sides of (85) with v, i.e.,

$$-\langle \nabla^2 u, v \rangle = \langle f, v \rangle. \tag{86}$$

To be more specific, we let $d = 2$ and take the inner product

$$\langle u, v \rangle = \iint_{\Omega} u(x, y) v(x, y) dx dy.$$

Then (86) becomes

$$-\iint_{\Omega} (u_{xx}(x, y) + u_{yy}(x, y)) v(x, y) dx dy = \iint_{\Omega} f(x, y) v(x, y) dx dy. \tag{87}$$

In order to be able to complete the derivation of the weak form we now assume that the space $\mathcal{U}$ of test functions is of the form

$$\mathcal{U} = \{v \ : \ v \in C^2(\Omega), \ v = 0 \text{ on } \partial\Omega\},$$

i.e., besides having the necessary smoothness to be a solution of (85), the functions also satisfy the boundary conditions.

Now we rewrite the left-hand side of (87):

$$\begin{aligned}\iint_\Omega (u_{xx}+u_{yy})\,v dxdy &= \iint_\Omega \left[(u_x v)_x + (u_y v)_y - u_x v_x - u_y v_y\right] dxdy \\ &= \iint_\Omega \left[(u_x v)_x + (u_y v)_y\right] dxdy - \iint_\Omega \left[u_x v_x - u_y v_y\right] dxdy. \quad (88)\end{aligned}$$

By using Green's Theorem (integration by parts)

$$\iint_\Omega (P_x + Q_y) dxdy = \int_{\partial\Omega} (Pdy - Qdx)$$

the first integral on the right-hand side of (88) turns into

$$\iint_\Omega \left[(u_x v)_x + (u_y v)_y\right] dxdy = \int_{\partial\Omega} \left(u_x v dy - u_y v dx\right).$$

Now the special choice of $\mathcal{U}$, i.e., the fact that v satisfies the boundary conditions, ensures that this term vanishes. Therefore, the weak form of (85) is given by

$$\iint_\Omega \left[u_x v_x + u_y v_y\right] dxdy = \iint_\Omega f v dxdy.$$

Another way of writing the previous formula is of course

$$\iint_\Omega \nabla u \cdot \nabla v dxdy = \iint_\Omega f v dxdy. \quad (89)$$

To obtain a numerical method we now need to require $\mathcal{U}$ to be finite-dimensional with basis $\{u_1, \ldots, u_n\}$. Then we can represent the approximate solution u^h of (85) as

$$u^h = \sum_{j=1}^{n} c_j u_j. \quad (90)$$

The superscript h indicates that the approximate solution is obtained on some underlying discretization of Ω with mesh size h.

Remark 1. In practice there are many ways of discretizing Ω and selecting $\mathcal{U}$.

(a) For example, regular (tensor product) grids can be used. Then $\mathcal{U}$ can consist of tensor products of piecewise polynomials or B-spline functions that satisfy the boundary conditions of the PDE.

(b) It is also possible to use irregular (triangulated) meshes, and again define piecewise (total degree) polynomials or splines on triangulations satisfying the boundary conditions.

(c) More recently, meshfree approximation methods have been introduced as possible choices for $\mathcal{U}$.

2. In the literature the piecewise polynomial approach is usually referred to as the *finite element method*.

3. The discretization of Ω will almost always result in a computational domain that has piecewise linear (Lipschitz-continuous) boundary.

We now return to the discussion of the general numerical method. Once we have chosen a basis for the approximation space $\mathcal{U}$, then it becomes our goal to determine the coefficients c_j in (90). By inserting u^h into the weak form (89), and selecting as trial functions v the basis functions of $\mathcal{U}$ we obtain a system of equations

$$\iint_\Omega \nabla u^h \cdot \nabla u_i dxdy = \iint_\Omega f u_i dxdy, \quad i = 1, \ldots, n.$$

Using the representation (90) of u^h we get

$$\iint_\Omega \nabla \left[\sum_{j=1}^{n} c_j u_j \right] \cdot \nabla u_i dxdy = \iint_\Omega f u_i dxdy, \quad i = 1, \ldots, n,$$

or by linearity

$$\sum_{j=1}^{n} c_j \iint_\Omega \nabla u_j \cdot \nabla u_i dxdy = \iint_\Omega f u_i dxdy, \quad i = 1, \ldots, n. \tag{91}$$

This last set of equations is known as the *Ritz-Galerkin method* and can be written in matrix form

$$Ac = b,$$

where the *stiffness matrix* A has entries

$$A_{i,j} = \iint_\Omega \nabla u_j \cdot \nabla u_i dxdy.$$

Remark 1. The stiffness matrix is usually assembled element by element, i.e., the contribution to the integral over Ω is split into contributions for each *element* (e.g., rectangle or triangle) of the underlying mesh.

2. Depending on the choice of the (finite-dimensional) approximation space $\mathcal{U}$ and underlying discretization, the matrix will have a well-defined structure. This is one of the most important applications driving the design of efficient linear system solvers.

Example One of the most popular finite element versions is based on the use of piecewise linear C^0 polynomials (built either on a regular grid, or on a triangular partition of Ω). The basis functions u_i are "hat functions", i.e., functions that are

piecewise linear, have value one at one of the vertices, and zero at all of its neighbors. This choice makes it very easy to satisfy the homogeneous Dirichlet boundary conditions of the model problem exactly (along a polygonal boundary).

Since the gradients of piecewise linear functions are constant, the entries of the stiffness matrix essentially boil down to the areas of the underlying mesh elements.

Therefore, in this case, the Ritz-Galerkin method is very easily implemented. We generate some examples with Matlab's PDE toolbox `pdetool`.

It is not difficult to verify that the stiffness matrix for our example is symmetric and positive definite. Since the matrix is also very sparse due to the fact that the "hat" basis functions have a very localized support, efficient iterative solvers can be applied. Moreover, it is known that the piecewise linear FEM converges with order $\mathcal{O}(h^2)$.

Remark 1. The Ritz-Galerkin method was independently introduced by Walther Ritz (1908) and Boris Galerkin (1915).

2. The finite element method is one of the most-thoroughly studied numerical methods. Many textbooks on the subject exist, e.g., "The Mathematical Theory of Finite Element Methods" by Brenner and Scott (1994), "An Analysis of the Finite Element Method" by Strang and Fix (1973), or "The Finite Element Method" by Zienkiewicz and Taylor (2000).

11.3 Optimality of the Ritz-Galerkin Method

How does solving the Ritz-Galerkin equations (91) relate to the solution of the strong form (85) of the PDE? First, we remark that the left-hand side of (89) can be interpreted as a new inner product

$$[u, v] = \iint_{\Omega} \nabla u \cdot \nabla v dx dy \tag{92}$$

on the space of functions whose first derivatives are square integrable and that vanish on $\partial\Omega$. This space is a *Sobolev space*, usually denoted by $H_0^1(\Omega)$.

The inner product $[\cdot, \cdot]$ induces a norm $\|v\| = [v, v]^{1/2}$ on $H_0^1(\Omega)$. Now, using this norm, the best approximation to u from $H_0^1(\Omega)$ is given by the function u^h that minimizes $\|u - u^h\|$. Since we define our numerical method via the finite-dimensional subspace $\mathcal{U}$ of $H_0^1(\Omega)$, we need to find u^h such that

$$u - u^h \perp \mathcal{U}$$

or, using the basis of $\mathcal{U}$,

$$\Big[u - u^h, u_i\Big] = 0, \quad i = 1, \ldots, n.$$

Replacing u^h with its expansion in terms of the basis of $\mathcal{U}$ we have

$$\left[u - \sum_{j=1}^{n} c_j u_j, u_i\right] = 0, \quad i = 1, \ldots, n,$$

or

$$\sum_{j=1}^{n} c_j \, [u_j, u_i] = [u, u_i], \quad i = 1, \ldots, n. \tag{93}$$

The right-hand side of this formula contains the exact solution u, and therefore is not useful for a numerical scheme. However, by (92) and the weak form (89) we have

$$\begin{aligned} [u, u_i] &= \iint_\Omega \nabla u \cdot \nabla u_i dxdy \\ &= \iint_\Omega f u_i dxdy. \end{aligned}$$

Since the last expression corresponds to the inner product $\langle f, u_i \rangle$, (93) can be viewed as

$$\sum_{j=1}^{n} c_j [u_j, u_i] = \langle f, u_i \rangle, \quad i = 1, \ldots, n,$$

which is nothing but the Ritz-Galerkin method (91).

The best approximation property in the Sobolev space $H_0^1(\Omega)$ can also be interpreted as an energy minimization principle. In fact, a smooth solution of the Poisson problem (85) minimizes the energy functional

$$E(u) = \frac{1}{2} \iint_\Omega \nabla^2 u dxdy - \iint_\Omega f u dxdy$$

over all smooth functions that vanish on the boundary of Ω. By considering the energy of nearby solutions $u + \lambda v$, with arbitrary real λ we see that

$$\begin{aligned} E(u + \lambda v) &= \frac{1}{2} \iint_\Omega \nabla(u + \lambda v) \cdot \nabla(u + \lambda v) dxdy - \iint_\Omega f(u + \lambda v) dxdy \\ &= \frac{1}{2} \iint_\Omega \nabla u \cdot \nabla u dxdy + \lambda \iint_\Omega \nabla u \cdot \nabla v dxdy + \frac{\lambda^2}{2} \iint_\Omega \nabla v \cdot \nabla v dxdy \\ &\quad - \iint_\Omega f u dxdy - \lambda \iint_\Omega f v dxdy \\ &= E(u) + \lambda \iint_\Omega \left[\nabla u \cdot \nabla v - f v \right] dxdy + \frac{\lambda^2}{2} \iint_\Omega \nabla^2 v dxdy \end{aligned}$$

The right-hand side is a quadratic polynomial in λ, so that for a minimum, the term

$$\iint_\Omega \left[\nabla u \cdot \nabla v - f v \right] dxdy$$

must vanish for all v. This is again the weak formulation (89).

A discrete "energy norm" is then given by the quadratic form

$$E(u^h) = \frac{1}{2} c^T A c - bc$$

where A is the stiffness matrix, and c is such that the Ritz-Galerkin system (91)

$$Ac = b$$

is satisfied.