Dr. Markus Faustmann
Paul Dunhofer
Institute of Analysis and Scientific Computing
Summer term 2025

TECHNISCHE UNIVERSITÄT WIEN

due on 14.03.2025

Scientific programming for interdisciplinary mathematics - Worksheet 1

Topics: convergence order, numerical differentiation, numerical methods for linear ODEs

Convergence order

We consider an abstract numerical method that approximates a quantity $y_\star \in \mathbb{R}$ by $\{y_h\}_{h>0} \subset \mathbb{R}$ such that

$$y_h \to y_\star \text{ for } h \to 0.$$

Here $h > 0$ is a discretization parameter that controls the accuracy of the procedure, i.e., smaller values of h lead to better approximations y_h. The method is said to have order of convergence $p > 0$ (or: convergence rate $p > 0$) if there exists a constant $C > 0$ such that

$$e_h \leq C\,h^p \quad \text{for all } h > 0, \text{ where } e_h := |y_\star - y_h|.$$

With the ansatz $e_h = C\,h^p$, the order of convergence can be expressed by the sequence $(p_h)_{h>0}$ with

$$p_h = \frac{\log(e_h/e_{h/2})}{\log(h/(h/2))} = \frac{\log(e_h) - \log(e_{h/2})}{\log(2)} \quad \text{for } h > 0.$$

Note that the calculation of p_h requires the availability of two iterates y_h, $y_{h/2}$ and, in particular, of the unavailable solution $y_\star$. However, one can show that the difference $\widetilde{e}_h := |y_{h/2} - y_h|$ behaves (asymptotically) like the error e_h. This motivates the formula for the *experimental* convergence order

$$\widetilde{p}_h = \frac{\log(\widetilde{e}_h) - \log(\widetilde{e}_{h/2})}{\log(2)} \quad \text{for } h > 0,$$

in which the (unknown) error e_h is approximated by $\widetilde{e}_h$. Note that the calculation of $\widetilde{p}_h$ requires the availability of three iterates y_h, $y_{h/2}$, and $y_{h/4}$, but not that of the exact solution $y_\star$.
In so-called double logarithmic plots (log-log plots) a logarithmic scaling is used for both coordinate axes, i.e., $\log(x)$ is plotted on the x-axis and $\log(y)$ on the y-axis.

Problem 1:
How are power functions of the form $e_h = C\,h^p$ represented in a log-log plot? How can you directly read the order p and the constant $C > 0$ from a log-log plot of $e_h = C\,h^p$?

Problem 2:
Consider a sufficiently smooth function $f \in C^k[x_0-\delta, x_0+\delta]$ for $x_0 \in \mathbb{R}$, $\delta > 0$ and $k \in \mathbb{N}$. Since the first derivative f' is often unknown, one must resort to numerical methods to approximate the value $f'(x_0)$ at the point x_0. One possibility is to use the one-sided difference quotient

$$D_h^1 f(x_0) := \frac{f(x_0+h)-f(x_0)}{h} \approx f'(x_0) \quad \text{for } 0 < h < \delta.$$

Write a MATLAB script implementing the one-sided difference quotient for $h \in \{2^{-n} : n = 0, \dots, N\}$ with $N = 10$. What rate of convergence do you observe for the error $e_h := |f'(x_0) - D_h^1 f(x_0)|$ for $f_1(x) := \exp(x)$, $f_2(x) := \sin(x)$, and $f_3(x) := x^{3/2}$ at $x_0 := 0$ for $h \to 0$? Represent the error graphically in a log-log plot, where you plot e_h over h.

Numerics of ODEs

The task is to compute a function $y\colon [0,T] \to \mathbb{R}^d$ from a given initial value $y_0 \in \mathbb{R}^d$ and a function $f \in C([0,T] \times \mathbb{R}^d; \mathbb{R}^d)$ determining the derivative y'. The corresponding *initial value problem* seeks $y \in C^1([0,T];\mathbb{R}^d)$ such that

$$y'(t) = f\big(t, y(t)\big) \quad \text{for all } t \in [0,T] \quad \text{and} \quad y(0) = y_0. \tag{1}$$

A numerical solution $(y_\ell)_{\ell=1}^L$ approximates $y(t_\ell) \approx y_\ell$ in certain nodes $0 = t_0 < t_1 < \dots < t_L = T$, e.g., in equidistant nodes, $t_{\ell+1} = t_\ell + h$ for all $\ell = 0, \dots, L-1$ and $h := T/L$. Taylor's theorem applied to the unknown function y and the ODE (1) shows, for $r, s \in [0,T]$ with $|s-r| \ll 1$, that

$$y(s) = y(r) + (s-r)\, y'(r) + \mathcal{O}(|s-r|^2) \approx y(r) + (s-r)\, f\big(r, y(r)\big). \tag{2}$$

The choice $s = t+h$ and $r = t$ leads us to the *explicit Euler method*

$$y_{\ell+1} = y_\ell + h\, f\big(t_\ell, y_\ell\big) \quad \text{for all } \ell = 0, \dots, L-1. \tag{3}$$

Alternatively, the choice $s = t$ and $r = t+h$ provides the *implicit Euler method*

$$y_{\ell+1} = y_\ell + h\, f\big(t_{\ell+1}, y_{\ell+1}\big) \quad \text{for all } \ell = 0, \dots, L-1. \tag{4}$$

Moreover, one may consider the implicit midpoint rule defined by

$$y_{\ell+1} = y_\ell + h\, f\Big(\frac{t_\ell + t_{\ell+1}}{2}, \frac{y_\ell + y_{\ell+1}}{2}\Big) \quad \text{for all } \ell = 0, \dots, L-1. \tag{5}$$

While $y_{\ell+1}$ in (3) can always be computed, (4)–(5) provide only an implicit determination of $y_{\ell+1}$ (since it is involved on both sides of the equality). For usual problems, the Banach fixed-point theorem proves that (4)–(5) have a unique solution $y_{\ell+1}$ if h is sufficiently small. A general *one-step method* is determined by the increment function $\Psi\colon [0,T] \times \mathbb{R}^d \times \mathbb{R}_+ \to \mathbb{R}^d$ in

$$y_{\ell+1} := y_\ell + h\, \Psi(t_\ell, y_\ell, h). \tag{6}$$

All these methods can be considered for variable step sizes $h_\ell := t_{\ell+1} - t_\ell$ instead of a uniform step size $h_\ell = h$.

Problem 3:
For $d = 1$, the scalar initial value problem (1) is called *linear* if

$$f(t, y) = f_1(t) + f_2(t)y \quad \text{with scalar functions } f_1, f_2 \colon \mathbb{R} \to \mathbb{R}. \tag{7}$$

Implement MATLAB functions for the one-step methods (3)–(5) to solve a linear scalar initial value problem. Adhere to the following signatures:

(a) $\texttt{y} = \texttt{explicitEuler}(\texttt{t}, \texttt{f1}, \texttt{f2}, \texttt{y0})$

(b) $\texttt{y} = \texttt{implicitEuler}(\texttt{t}, \texttt{f1}, \texttt{f2}, \texttt{y0})$

(c) $\texttt{y} = \texttt{implicitMidpoint}(\texttt{t}, \texttt{f1}, \texttt{f2}, \texttt{y0})$

The arguments consist of the row vector of time steps $\texttt{t} = (t_0, \dots, t_L)$ (not necessarily equidistant), the function handles $\texttt{f1}, \texttt{f2}$ determining the right-hand side f, and the initial value $\texttt{y0}$. The output $\texttt{y} = (y_0, \dots, y_L)$ is the row vector containing the approximations to the solution y in the time steps $\texttt{t}$, i.e., $y_\ell \approx y(t_\ell)$.

Problem 4:
Consider the scalar initial value problem

$$y'(t) = -y(t) \quad \text{for all } t \in [0, 1] \quad \text{and} \quad y(0) = y_0 := 1. \tag{8}$$

Note that (8) is linear (7) with $f_1(t) := 0$ and $f_2(t) := -1$ for all $t \in [0, 1]$. Compute approximations to the solution with all three functions $\texttt{y} = \texttt{method}(\texttt{t}, \texttt{f1}, \texttt{f2}, \texttt{y0})$ from Problem 3. Use equidistant time steps with step size $h = 1/L$. The exact solution to (8) reads $y(t) = \exp(-t)$ and enables the computation of the exact error $e_h := \max\limits_{\ell=1,\dots,L} |y(t_\ell) - y_\ell|$ in the maximum norm.

(a) Plot the exact solution y and the three approximations $(y_\ell)_\ell$ in dependence of t into one plot.

(b) Plot the error e_h in dependence of the step size $h \in \{2^{-n} \ : \ n = 1, \dots, N\}$ with $N = 10$ and compare the convergence rates for the different methods.

Problem 5:
Modify your three functions from Problem 3 for the solution of systems of linear initial value problems (1) with f satisfying (7) for $f_1 : \mathbb{R} \to \mathbb{R}^d$ and $f_2 : \mathbb{R} \to \mathbb{R}^{d\times d}$. The signatures of the functions should remain the same as in Problem 3. For the row vector $\texttt{t} = (t_0, \dots, t_L) \in \mathbb{R}^{L+1}$ as input, the output $\texttt{y} = (y_0, \dots, y_L) \in \mathbb{R}^{d\times(L+1)}$ should provide the columns $\texttt{y}(:, \ell + 1) = y_\ell \approx y(t_\ell) \in \mathbb{R}^d$. Ahead of the implementation, derive mathematically what has to be done. Deriving the mathematical formula for the implementation, what is the difference in the application of the explicit and implicit methods?

Problem 6:
Consider the linear system of ordinary differential equations

$$\begin{Bmatrix} y_1' = -y_2 \\ y_2' = y_1 \end{Bmatrix} \quad \text{with initial values} \quad \begin{Bmatrix} y_1(0) = 1 \\ y_2(0) = 0 \end{Bmatrix} \tag{9}$$

Apply the three methods (3)–(5) using the functions from Problem 5 to solve the system (9). What are the functions f_1 and f_2 in this example? The exact solution to (9) reads $y(t) := (\cos(t), \sin(t))^\top$ and enables the computation of the exact error e_h as in Problem 4 (with the absolute value $|\cdot|$ replaced by the maximum norm $\|\cdot\|_\infty$ on $\mathbb{R}^d$).

(a) Plot the error $e_h := \max_{\ell=1,\dots,L} \|y(t_\ell) - y_\ell\|_\infty$ in dependence of the step size $h \in \{2^{-n} : n = 1, \dots, N\}$ with $N = 10$ and compare the convergence rates for the different methods.

(b) Plot the exact and the discrete solution in a phase field plot, i.e., the component $y_{\ell,1}$ (on the y-axis) in dependence of $y_{\ell,2}$ (on the x-axis). What does the exact solution look like in the phase field plot? What is the difference between the three one-step methods? Use a larger step size, e.g., $h = 0.1$, to see the differences more clearly.

Runge–Kutta methods

The most important class of one-step methods are the so-called Runge–Kutta methods. Let $A \in \mathbb{R}^{m\times m}$ and $b, c \in \mathbb{R}^m$ with $0 \le c_1 \le c_2 \le \dots \le c_m \le 1$. An *$m$-stage Runge–Kutta method* is a one-step method in the sense of (6) with the increment function

$$\Psi(t, y, h) := \sum_{j=1}^{m} b_j k_j, \tag{10}$$

where the so-called *stages* satisfy the implicit conditions

$$k_j = f\Big(t + c_j h,\, y + h \sum_{\ell=1}^{m} A_{j\ell} k_\ell\Big) \quad \text{for all } j = 1, \dots, m. \tag{11}$$

The method is called *explicit* if A is a strictly lower triangular matrix, i.e., $A_{j\ell} = 0$ for all $j \le \ell$. Otherwise, it is called *implicit*. Usually, a Runge–Kutta method is written in terms of its *Butcher tableau* $\begin{array}{c|c} c & A \\ \hline & b \end{array}$, where zero entries of A are usually neglected. As for the implicit methods (4)–(5), one can show that for usual problems the implicit formulation (11) admits a unique solution $\boldsymbol{k} = (k_1, \dots, k_m) \in \mathbb{R}^m$ by means of the Banach fixed-point theorem.

Problem 7:
Implement a MATLAB function

$$\texttt{y} = \texttt{explicitRungeKutta}(\texttt{A}, \texttt{b}, \texttt{c}, \texttt{t}, \texttt{f1}, \texttt{f2}, \texttt{y0})$$

for an *explicit* m-step Runge–Kutta method given by the *strictly lower triangular* matrix $\texttt{A} \in \mathbb{R}^{m\times m}$ and $\texttt{b}, \texttt{c} \in \mathbb{R}^m$, applied to a scalar and linear initial value problem (see Problem 4).

(a) Which of the three methods (3)–(5) are Runge–Kutta methods and which can be implemented by your `explicitRungeKutta` function? What are the respective Butcher tableaus?

(b) Compare the new function `explicitRungeKutta.m` to the direct implementation from Problem 3.

Problem 8:
Extend your implementation of *explicit* Runge–Kutta methods from Problem 7 to general linear initial value problems, i.e., general $d \geq 1$; cf. Problem 5.

Problem 9:
Use your implementation from Problem 8 to solve the initial value problem (9) from Problem 6. Test the classical RK4 method given by the *Butcher tableau*

$$\begin{array}{c|c} c & A \\ \hline & b \end{array} \quad = \quad \begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

(a) Compute the error and compare the convergence rate with the results from Problem 6 (a). What convergence rate do you observe?

(b) Plot the solution into the phase field plot from Problem 6 (b).

Problem 10:
Implement a MATLAB function `y = implicitRungeKutta(A, b, c, t, f1, f2, y0)` for arbitrary (implicit) Runge–Kutta methods for a *scalar and linear* initial value problem (see Problem 4).

Hint: Proceed as for the implicit Euler method (4) in Problem 3 and write the stage-conditions (11) as an $m \times m$ linear system with solution $\boldsymbol{k} = (k_1, \ldots, k_m)^\top \in \mathbb{R}^m$.

Problem 11:
Consider the 2-stage and 3-stage Gauss–Runge–Kutta methods

$$\begin{array}{c|c} c & A \\ \hline & b \end{array} \quad = \quad \begin{array}{c|cc} (3-\sqrt{3})/6 & 1/4 & (3-2\sqrt{3})/12 \\ (3+\sqrt{3})/6 & (3+2\sqrt{3})/12 & 1/4 \\ \hline & 1/2 & 1/2 \end{array}$$

$$\begin{array}{c|c} c & A \\ \hline & b \end{array} \quad = \quad \begin{array}{c|ccc} 1/2-\sqrt{15}/10 & 5/36 & 2/9-\sqrt{15}/15 & 5/36-\sqrt{15}/30 \\ 1/2 & 5/36+\sqrt{15}/24 & 2/9 & 5/36-\sqrt{15}/24 \\ 1/2+\sqrt{15}/10 & 5/36+\sqrt{15}/30 & 2/9+\sqrt{15}/15 & 5/36 \\ \hline & 5/18 & 4/9 & 5/18 \end{array}$$

What convergence rates do you observe for the initial value problem from Problem 4.

Take-home message

Explicit Runge–Kutta methods (like the explicit Euler method) to solve ODEs

$$y' = f(t, y(t))$$

can easily be implemented for arbitrary (nonlinear) f and any dimension $d \geq 1$. Implicit Runge–Kutta methods (like the implicit Euler method or the midpoint scheme) lead to nonlinear systems of equations that must be solved to compute the stages. If f is linear in the sense of (7), then

- 1-step Runge–Kutta methods for arbitrary $d \geq 1$,
- m-step Runge–Kutta methods for $d = 1$

require the solution of one linear system per time step. Actually, the same holds indeed for arbitrary $m \geq 1$ and $d \geq 1$.