Numerical approximation of solutions to initial values problems

Chapter 7.1, 7.2 and 7.3.

Numerical approximation of solutions to initial values problems

Chapter 7.1, 7.2 and 7.3.

This and next lectures are about ordinary differential equations (ODEs) of the form

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big),$$

where f is a scalar function of two variables t and x.

Numerical approximation of solutions to initial values problems

Chapter 7.1, 7.2 and 7.3.

This and next lectures are about ordinary differential equations (ODEs) of the form

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big),$$

where f is a scalar function of two variables t and x.

In next week: $\boldsymbol{f}$ can be a vector function of t and $\boldsymbol{x}$-vector.

Numerical approximation of solutions to initial values problems

Chapter 7.1, 7.2 and 7.3.

This and next lectures are about ordinary differential equations (ODEs) of the form

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big),$$

where f is a scalar function of two variables t and x.

In next week: $\boldsymbol{f}$ can be a vector function of t and $\boldsymbol{x}$-vector.

The numerical approximation $(x_1, x_2, \ldots, x_n)$ is

- an approximation of the unknown solution $x(t)$, and
- only given at discrete t-values $t_i = t_0 + ih, \quad i = 1, \ldots, n$:

$$x_0 \approx x(t_0), x_1 \approx x(t_1), \ldots, x_n \approx x(t_n).$$

Partial Differential Equations (with more independent variables $t, y, \ldots$) are introduced e.g. in the course 01418 and their numerical methods in 02686, 02687, 02689.

Higher-order differential equation $\longrightarrow$ system of first order

Numerical solvers are **only** for systems of first-order differential equations.

A single differential equation of order n,

$$\frac{d^n x}{dt^n} = f\left(t, x, \frac{dx}{dt}, \ldots, \frac{d^{(n-1)}x}{dt^{(n-1)}}\right),$$

can be turned into a system of n first-order equations of the form

$$\frac{d\boldsymbol{z}(t)}{dt} = \boldsymbol{f}\left(t, \boldsymbol{z}(t)\right).$$

It requires n auxiliary conditions (e.g. initial conditions) in order to specify the solution precisely.

Initial values problems

Example:

$$\frac{dx}{dt} = -3x$$

Initial values problems

Example:

$$\frac{dx}{dt} = -3x$$

$x(t) = ke^{-3t}$ for any $k \in \mathbb{R}$.

With an extra condition, e.g. $x(0) = 7$, a unique solution is determined:

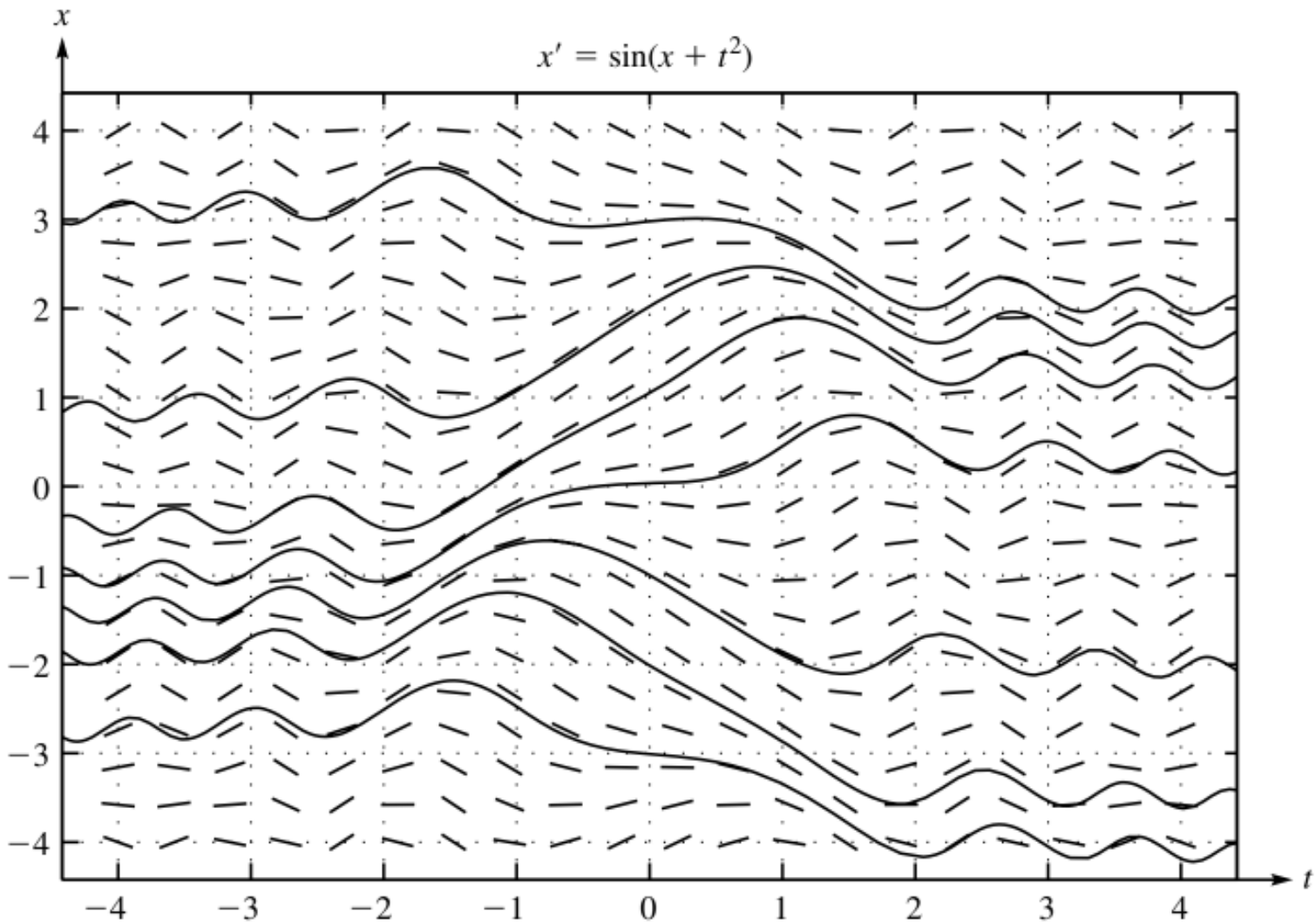
$$x(t) = 7e^{-3t}, \quad t \in \mathbb{R}.$$

Initial-value problems (this week):

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big), t > t_0, \qquad \text{with } x(t_0) = x_0.$$

Vector fields

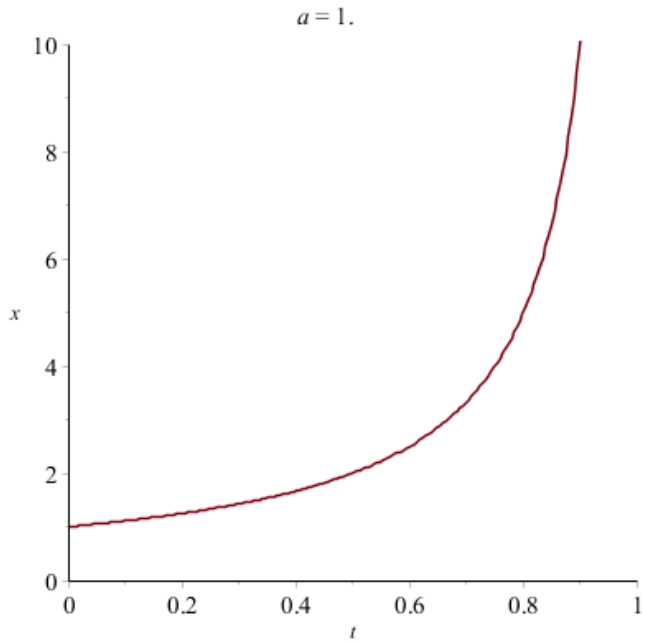
$$x' = f(x,t) = \sin(x + t^2)$$



Existence and uniqueness of solutions

Theorem:

If f is differentiable, then the initial value problem has a unique solution for $x(t)$ defined for $t \in (t_0, t_0 + \epsilon)$.



Existence and uniqueness of solutions

Theorem:

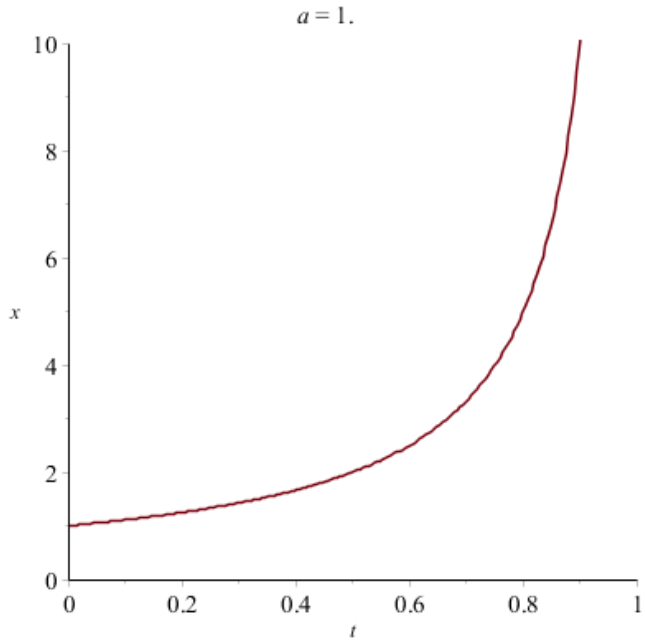
If f is differentiable, then the initial value problem has a unique solution for $x(t)$ defined for $t \in (t_0, t_0 + \epsilon)$.

Example: For $a > 0$ the solution to

$$\frac{dx}{dt} = x^{1+a}, \quad x(0) = 1$$

is

$$x(t) = \frac{1}{(1 - at)^{\frac{1}{a}}}.$$



Existence and uniqueness of solutions

Theorem:

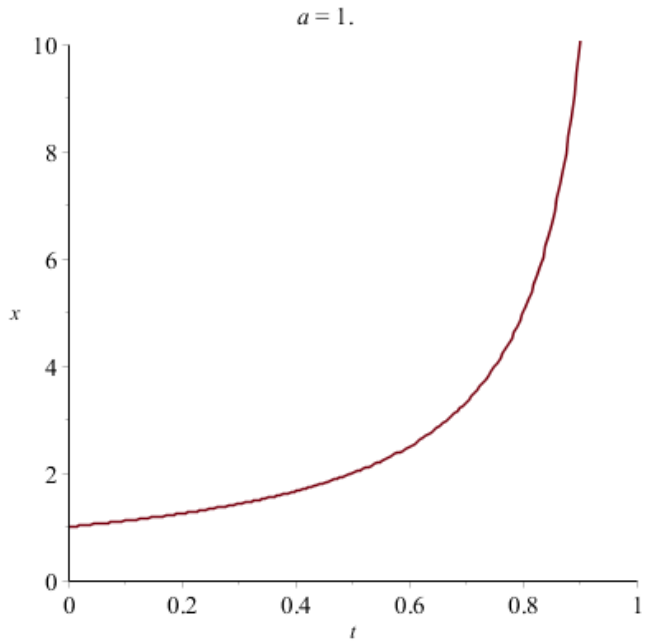
If f is differentiable, then the initial value problem has a unique solution for $x(t)$ defined for $t \in (t_0, t_0 + \epsilon)$.

Example: For $a > 0$ the solution to

$$\frac{dx}{dt} = x^{1+a}, \quad x(0) = 1$$

is

$$x(t) = \frac{1}{(1 - at)^{\frac{1}{a}}}.$$



But the solution exists only as $t < \frac{1}{a}$.

Note: Numerical studies requires an idea about which time interval makes sense to look for a solution in.

Euler's method – one step

Consider the initial-value problem

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big), \;\; x(a) = x_a.$$

The second order Taylor polynomium for x around t is

$$x(t+h) \approx x(t) + hx'(t) = x(t) + hf\Big(t, x(t)\Big).$$

Plugging in $t = t_0$ we approximate $x(t+h)$ by Euler's formula

$$x(t+h) \approx x_1 = x_0 + hf\Big(t, x_0\Big).$$

Note that x_1 is obtained from x_0 by following the slope field $f(t, x)$.

Euler's method – more steps

Now split the interval $[a, b]$ into n subintervals with size $h = (b - a)/n$. Euler's method starts from $x(a) = x_a$ and after n steps

$$\begin{aligned} x_0 &= x(a) = x_a \rightarrow \\ x_1 &\approx x(a + h) \rightarrow \\ x_2 &\approx x(a + 2h) \rightarrow \\ &\cdots \rightarrow \\ x_n &\approx x(a + nh) = x(b) \end{aligned}$$

ending with an approximation of $x(b)$.

- Euler's method is not commonly used in practice. But it's good for illustration.
- The following methods can be considered as generalized Euler's method.
- Have a look at `MyEuler.py` and `TestEuler.py`

Taylor series methods

Euler's method only uses $f\big(t, x(t)\big) = x'(t)$.
If we have higher order derivatives, e.g.

$$x''(t) = \frac{d\, f\big(t, x(t)\big)}{dt} = f'_t(t,x) + f'_x(t,x)x' = \frac{df}{dt}\big(t, x, x'\big),$$
$$x'''(t) = \cdots = \frac{d^2 f}{dt^2}\big(t, x, x', x''\big),$$

then based on Taylor series we can obtain better approximation

Taylor series methods of the first/second/third order are:

```
dx=f(t,x)                                  % given by user
d2x=dfdt(t,x,dx)                           % given by user
d3x=d2fdt2(t,x,dx,d2x)                     % given by user
x=x+h*dx                          % Euler's method; the 1st order
x=x+h*(dx+1/2*h*(d2x))                          %  2nd order
x=x+h*(dx+1/2*h*(d2x+1/3*h*(d3x)))              % 3rd order
```

Taylor series methods

When we use Taylor series method of order k, only the terms up to order k in Taylor series of $x(t)$ are included, which leads to a *local truncation error*

$$\frac{1}{(k+1)!}x^{(k+1)}(\xi)h^{k+1}, \quad \text{some } \xi \in (a,b)$$

in each iteration.

But how is the local error accumulated to a global error?

We will give an estimate for Euler's method later today, and show a numerical experiment.

Taylor series methods

- It's easy to understand
- It's easy to implement.
- But ....

Taylor series methods

- It's easy to understand
- It's easy to implement.
- But .... the user need provide

$$f(t,x), \frac{df}{dt}(t,x,x'), \frac{d^2f}{dt^2}(t,x,x',x''), \ldots$$

- Then some other tools can be needed, e.g.
  - symbolic differentiation (like Maple) or
  - automatic differentiation (a program that "differentiates another program")
- Methods that **only** use $f(t,x)$ are preferred.

Heun's method and midpoint method

Heun's method and midpoint method

Both methods shown on the board can be written in the form:

$$\begin{aligned} K_1 &= f(t, x) \\ K_2 &= f(t + \alpha h, x + \beta h K_1) \\ x(t + h) &= x(t) + h * \left(\omega_1 K_1 + \omega_2 K_2\right) \end{aligned}$$

Such methods are called Runge-Kutta methods.

NB: In the book, the notations are defined as

$$K_1 = h\, f(t, x) \text{ and } K_2 = h\, f(t + \alpha h, x + \beta K_1).$$

Error estimation for Runge-Kutta methods of order 2

In `RungeKuttaOrden2Fejlestimat.mw` it shows all terms up to second order in the Taylor series of exact solution equals the Runge-Kutta approximation if the following conditions are satisfied:

$$1 = \omega_1 + \omega_2$$
$$\frac{1}{2} = \omega_2 \alpha$$
$$\frac{1}{2} = \omega_2 \beta$$

Hence, the error in each iteration is $\mathcal{O}(h^3)$ or $\mathcal{O}(1/n^3)$.

That is, the local error is proportional to h^3 or to $1/n^3$ for a small value of h or a large value of n.

For Euler's method, the local error is $\mathcal{O}(h^2)$.

Global error estimation for Runge-Kutta methods of order 2

Initial values problem:

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big), \;\; x(a) = x_a, \;\; \text{for} \;\; t \in [a, b].$$

- Split the interval $[a, b]$ into n subintervals with size $h = (b - a)/n$ and use a Runge-Kutta method n times.
- This gives the estimation x_i of $x(t_i)$ at the discrete time $t_i = a + ih$.

$$x(a) = x_0 \to x_1 \to x_2 \to \cdots \to x_n \approx x(b)$$

- There are n local errors, and every error is 'proportional' to h^3 and to $1/n^3$.
- Bound: The global error is 'proportional' to h^2 and to $1/n^2$.
- For Euler's method, the local error is $\mathcal{O}(h^2)$. Then is the global error $\mathcal{O}(h^1)$?

Let's try!

Example

Consider the initial values problem

$$\frac{dx}{dt} = t^2 - 2x, \;\; x(0) = x_0 = 1, \;\; t \in [0, 1],$$

which has the exact solution $x(t) = \frac{1}{4} + t\left(-\frac{1}{2} + \frac{1}{2}t\right) + \frac{3}{4}e^{-2t}$.

Here we assess the accuracy of the numerical solution by checking the relative error at the end point in the interval

$$\left| \frac{x_{\text{numerical}}(1) - x(1)}{x(1)} \right| = \left| \frac{x_n - x(1)}{x(1)} \right|.$$

NB: You also can use other error measures.

Example

The 2nd-order methods need two function evaluations in each iteration; so we count the total number of function evaluations, n_f, and compare the relative error:

n_f	n_{order1}	n_{order2}	*Euler*	*Midpoint*	*Heun*
10	10	5	0.1231	0.0367	0.0519
20	20	10	0.0606	0.0079	0.0113
40	40	20	0.0301	0.0018	0.0027
80	80	40	0.0150	0.0004	0.0006

The 2nd-order methods give better accuracy with the same effort!

For Euler's method the global error is reduced to half, when n is doubled, while for 2nd-order methods the global error is divided by 2^2.

Their global errors are $\mathcal{O}(1/n)$ and $\mathcal{O}(1/n^2)$, respectively – equals to the sum of the local errors.

Runge-Kutta methods of order 4

One commonly used 4th-order method is:

$$x(t+h) = x(t) + \frac{h}{6}\Big(K_1 + 2K_2 + 2K_3 + K_4\Big)$$

where

$$\begin{aligned} K_1 &= f(t, x) \\ K_2 &= f(t + {}^{h}/_{2}, x + {}^{h}/_{2}K_1) \\ K_3 &= f(t + {}^{h}/_{2}, x + {}^{h}/_{2}K_2) \\ K_4 &= f(t + h, x + hK_3). \end{aligned}$$

NB: Again, in the book it defines $K_1 = hf(t, x)$..., but we follow the same notations as in Chapter 7.4.

Runge-Kutta methods of order 4

The derivation of the Runge-Kutta formulas of order 4 follows that the solution at $x(t+h)$ agrees with the Taylor expansion up to and including the term in h^4. Hence, the local error is $\mathcal{O}\left(1/n^5\right)$.

Therefore, the global error is $\mathcal{O}\left(1/n^4\right)$.

In general, Runge-Kutta methods of order k have local error $\mathcal{O}\left(1/n^{(k+1)}\right)$ and global error $\mathcal{O}\left(1/n^k\right)$.

Adaptive methods

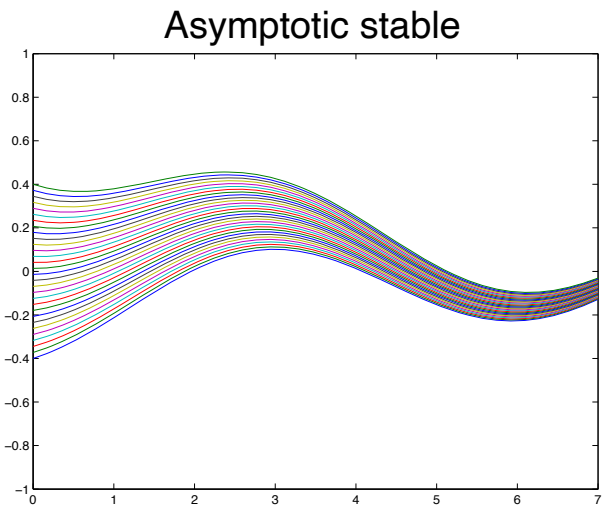
Similar as in integration, it's interesting to develop new methods such that the step size depending on the right-hand side of ODE.

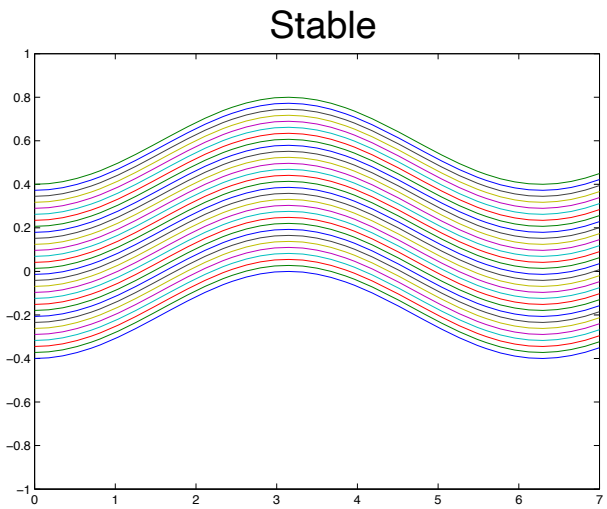
There is a widely used algorithm which is also implemented in the scipy.integrate package: `solve_ivp`. By default, it uses RK45, which based on the computation of 6 derivatives, calculates both a 4th and a 5th order Runge-Kutta method. It also gives an error estimate that can be used for an adaptive procedure. See Chapter 7.3.

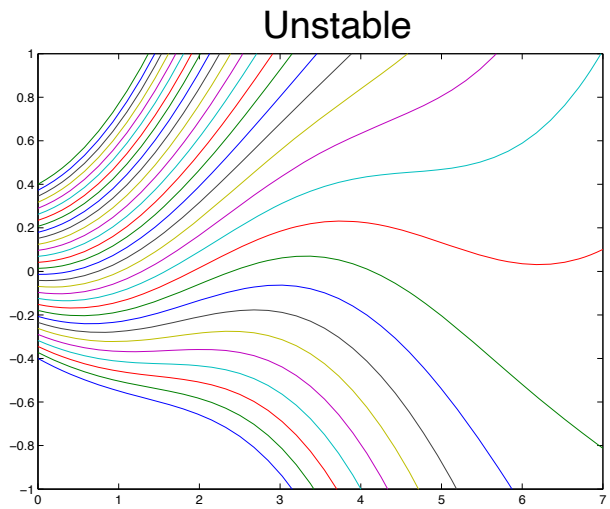
Runge-Kutta methods are more popularly used, because the change of the step size is trivial. In the procedure, step i only depends on step $i-1$ and the step size.

More efficient "multistep" algorithms can be done by changing step i depending on several previous steps. Commonly used in molecular dynamics and game physics.

Stability Analysis







A solution of an ordinary differential equation is called

- Stable: if the solutions of the initial-value problem with small changes on the initial value stay close to the original solution.
- Asymptotic stable: if it's stable and the solutions from small changes always converge to the original solution.
- Unstable: if the solutions from small changes diverge from the original solution.

Stability analysis

- Even for a stable solution, a numerical ODE solver can become unstable, if the step size is chosen too large.
- Stiff systems where different phenomena occur on different time scales require other solvers.....
- This is out of scope for this course, but you can learn more in e.g. Course 02686 & 02687 (in total 10 ECTS only on numerical methods for ODE's and PDE's)

Next week

In Chapter 7.4, we consider the ODE systems. A single differential equation

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big)$$

is extended to a system of differntial equations

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big).$$

Instead of solving a scalar x, now we need solve a vector $\boldsymbol{x}$, and f is replaced by a vector function $\boldsymbol{f}$ in the same dimension as $\boldsymbol{x}$. The algorithms are similar, but now scalars are replaced by vectors.

So Chapter 7.4 can be quickly gone through. After that, we will deal with boundary-value problems, i.e. Chapter 11.1, and here we need both interpolation and equation solvers.