Numerical methods
for solving 1st order, ordinary differential equations (ODEs) with initial conditions

common structure: \( \frac{dy}{dt} = f(t, y) \) with initial value \( y(t_0) = y_0 \)

Euler method:

Taylor expansion: \( y(t + \Delta t) = y(t) + \frac{dy}{dt} \Delta t + O((\Delta t)^2) \)

insert ODE \( \frac{dy}{dt} = f(t, y) \) \( \Rightarrow \) \( y(t + \Delta t) = y(t) + f(t, y) \Delta t + O((\Delta t)^2) \)

introduce discrete times \( t(i) \) with fixed time steps \( \Delta t \)
\( t(i + 1) = t(i) + \Delta t \) with \( t(1) = t_0 \) (initial time), \( t(N + 1) = t_f \) (final time) and \( y(i) = y(t(i)) \)

Euler method: \( y(i + 1) = y(i) + f(t(i), y(i)) \Delta t \) with \( y(1) = y_0 \)

Note: The Euler method is the simplest and “fastest” method (only one evaluation of \( f \) per time step) but at the price of accuracy and precision.
The simplest improvement is going to smaller time steps (error \( \sim (\Delta t)^2 \) ).
However, errors accumulate quickly and the method is not necessarily stable (see pendulum).

2nd order Runge-Kutta method with fixed step size

Remember: the centered difference approximation to the derivative is an order of \( \Delta t \) better than the forward difference approximation.

middle of the time interval: \( t_{\text{mid}}(i) = t(i) + \frac{1}{2} \Delta t \)

with centered difference approximation: \( y(i + 1) \cong y(i) + \Delta t \frac{dy}{dt} \bigg|_{t_{\text{mid}}(i)} = y(i) + \Delta t f(t_{\text{mid}}(i), y_{\text{mid}}) \)

How do we find \( y_{\text{mid}}(i) \)?

one answer: \( y_{\text{mid}} \cong y(i) + \frac{\Delta t}{2} \frac{dy}{dt} \bigg|_{t(i)} = y(i) + \frac{\Delta t}{2} f(t(i), y(i)) \)

2nd - order Runge - Kutta method
\( y(i + 1) = y(i) + \Delta t f(t(i) + \frac{1}{2} \Delta t, y(i) + \frac{1}{2} \Delta t f(t(i), y(t(i)))) \) with \( y(1) = y_0 \)

Note: for each time step, two function calls have to be evaluated (in contrast to one function call for the Euler method) but the error \( \sim (\Delta t)^3 \) is an order of \( \Delta t \) smaller than for the Euler method.
The Runge-Kutta method is based on a Taylor series expansion of \( y(t + \Delta t) \) and can be systematically improved by keeping higher order terms at a cost of more time (see Numerical Recipes)
4th order Runge-Kutta method with fixed step size

The most widely used fixed step-size Runge Kutta method is of 4th order

Let
\[ k_1 = \Delta t f(t(i), y(t(i))) \]
\[ k_2 = \Delta t f(t(i) + \frac{1}{2} \Delta t, y(t(i)) + \frac{1}{2} k_1) \]
\[ k_3 = \Delta t f(t(i) + \frac{1}{2} \Delta t, y(t(i)) + \frac{1}{2} k_2) \]
\[ k_4 = \Delta t f(t(i) + \Delta t, y(t(i)) + k_3) \]

2nd - order Runge - Kutta method
\[ y(i+1) = y(i) + k_2 + O(\Delta t^3) \]

4th - order Runge - Kutta method
\[ y(i+1) = y(i) + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) + O(\Delta t^5) \]

Note: when you use a fixed step-size method, always make sure in the end that your results do not change significantly with step size.

Note: going to higher order and smaller step-sizes to improve accuracy can be very time consuming and may not be the best approach → investigate the source of the problem

Adaptive step-size methods

Consider a function \( y(t) \) like this:

It is desirable to have: (why?)
- small steps in the region of large variation
- large steps in the region of small variation

⇒ adapt the step size of a Runge-Kutta method as you are calculating the solution

Idea of a 4th/5th order method:

For a given step size \( \Delta t \)
do a single step in 5th order approximation \( \Rightarrow y(i+1) \)
do the same step in 4th order approximation \( \Rightarrow y^*(i+1) \)

Then \( \Delta y = |y(i+1) - y^*(i+1)| \) is a measure for the error, which is \( O(\Delta t)^5 \)

Hence, we can determine a new step size \( \Delta t \) that matches our tolerance for \( \Delta y \)
if \( \Delta y \) is too large ⇒ reduce the step size \( \Delta t \) and try again
if \( \Delta y \) is too small ⇒ increase the step size \( \Delta t \) and try again
There are several ways to specify the tolerance:

1. **by relative error**
   When would this be a problem? \( \frac{\Delta y}{y(t)} < \varepsilon_{relative} = \text{RelTol} \) in Matlab

2. **by absolute error**
   What has to be considered here? \( |\Delta y| < \varepsilon_{absolute} = \text{AbsTol} \) in Matlab

3. **by cumulative error**
   - if conserved quantities are known (for example energy or momentum) one can adjust the step size to keep the conserved quantities within tolerance
   - this requires special programming for each particular problem, but is very powerful when available

Matlab provides two ODE solvers based on adaptive step-size Runge-Kutta methods ode23 (2nd/3rd order method) and ode45 (4th/5th order method)

% calculate a solution to the nuclear decay problem from the built-in ode45 solver
options=odeset('RelTol',1.e-6,'AbsTol',1.e-6);
[tt,yode45]=ode45('f1nuc',t,y0,options);

solution, rhs of \( \frac{dy}{dt} \), times, initial vector \( y(t) \), dy/dt, vector condition