Ordinary differential equations, part 1
Scientific Computing, bridging course

ODEs introduction

- Example
\[ \begin{align*}
v(t) &= g - \frac{c}{m} v \\
v(0) &= v_0
\end{align*} \]
describes the velocity of a skydiver with mass \( m \), acceleration due to gravity \( g \), drag coefficient \( c \).
- Here, right-hand-side
\[ f(t, v) = g - \frac{c}{m} v, \quad g, c, m \] parameters

Interpretation: Change in velocity at time \( t \) is proportional to velocity at time \( t \) plus acceleration due to gravitation.
- The initial value must be known, e.g., velocity at time 0, \( v(0) \), otherwise no unique solution

From computer lab

- ODEs written on the form
\[ \begin{align*}
\frac{dy}{dx} &= f(x, y) \\
y(a) &= y_0
\end{align*} \]
- Right-hand-side \( f(x, y) \) defined in a function and sent as an inparameter to Matlab's solvers

```matlab
function yout = rhsODE(t, y)
yout = ...; % beräkning av högerledet
```

```matlab
tspan = [t0 t_end]; % interval
y0= [...] % initial values
[t,y]=ode45(rhsODE, tspan, y0);
```
From computer lab

- Matlab's ODE solvers use rhs-functionen internally, once every time step
- No principal difference between solving one equation or a system of equations
- Important numerical methods: Euler's method, Heun's method, Classical Runge-Kutta
- Classical Runge-Kutta more accurate, Euler's method not so accurate
- Accuracy depend on step length \( h \) (discretization parameter)
- Matlab's solvers find \( h \) automatically, and vary it over the interval (given a certain tolerance)

From computer lab

Some questions:
- The principles behind the methods? (The lab showed they're based on slopes at different points)
- Why is Heun and Classical Runge-Kutta more accurate than Euler?
- In what way does the accuracy depend on step length \( h \)?
- How can Matlab's solvers find step length automatically and keep a certain tolerance, without knowing the true solution?
- Other questions?

About ODEs

- Two main types
  - Initial value problems
    \[
    \begin{align*}
    \dot{y}(t) &= f(t,y), & t \geq a \\
    y(a) &= y_0
    \end{align*}
    \]
  - Boundary value problems
    \[
    \begin{align*}
    y(x) &= f(x,y), & a \leq x \leq b \\
    y(a) &= y_a \\
    y(b) &= y_b
    \end{align*}
    \]
- Vibrating strings, beam deflection, ...

About ODEs

- Here only initial value problems
- Different types depending on the right-hand-side
  - Linear with constant coefficients
    \[ f(t,y) = y + t \]
  - Linear with variable coefficients
    \[ f(t,y) = \sin(t)y + t \]
  - Non-linear
    \[ f(t,y) = y^2 + t \]

About ODEs

- Often systems av ODEs, such as
  \[
  \begin{align*}
  \dot{y}_1 &= \dot{y}_2 \\
  \dot{y}_2 &= \dot{y}(y_1, y_2, y_3) = f(t,y)
  \end{align*}
  \]
- Can be written as vectors
  \[
  \begin{pmatrix}
  \dot{y}_1 \\
  \dot{y}_2 \\
  \end{pmatrix} = \begin{pmatrix} y_1 & y_2 \end{pmatrix} \Rightarrow \dot{y}(t,y) = f(t,y)
  \]

Hence, same form as scalar ODEs

About ODEs

- Numerical methods solve ODEs on the form
  \[ \dot{y} = f(t,y) \]
- What can you do if you get an higher order ODE: \( y'' + cy' + g \sin(y) = 0 \) \( \Rightarrow y'' = -cy' - g \sin(y) \)

Rewrite:

Let \( u_1 = y \) \( \Rightarrow u_1' = y' \)
\( u_2 = y' \) \( \Rightarrow u_2' = y'' = -cu_2 - g \sin(u_1) \)
\( \Rightarrow u' = f(t,u) \), i.e. the right form, now you can apply the ODE solver on the system. If higher derivatives than 2nd, just repeat the process.
About ODEs

- Example: Solve Van der Pohl’s eq. in Matlab:

\[ y'' - \lambda (1 - y^2)y' + y = 0 \implies y'' = \frac{\lambda (1 - y^2)y' - y}{y(0) = 2, y'(0) = 0} = f(t, y', y) \]

Rewrite to system of 1st order ODEs:

\[ \begin{align*}
  u_1 &= y \implies u_1' &= y' \\
  u_2 &= y' \implies u_2' &= u_2 - u_1
\end{align*} \]

and in vector form

\[ \begin{pmatrix} u_1' \\ u_2' \end{pmatrix} = \begin{pmatrix} u_2 \\ \lambda (1 - u_1^2)u_2 - u_1 \end{pmatrix}, u(0) = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \]

Now on the form \( u' = f(t, u) \), i.e. right form for Matlab. The looked after solution \( y \) can be found in first entry of vector \( u \).

ODEs in Matlab

- Solve in Matlab. Define rhs in a Matlab function

\[
\text{function } u_{\text{out}} = \text{vdp}(t, y, \lambda) \%
\text{ Van der Pohl equation}
\]

\[
\begin{align*}
  &\text{u}_{\text{out}} = [u(2); \\
  &\lambda \cdot (1-u(1)^2) \cdot u(2) - u(1)];
\end{align*}
\]

Note, \( u \) is a vector where you find the solution in first position \( u(1) \)

\[ u' = \begin{pmatrix} u_1' \\ u_2' \end{pmatrix} = \begin{pmatrix} u_2 \\ \lambda (1-u_1^2)u_2 - u_1 \end{pmatrix} \]

And the result is...

Solution to Van der Pol’s eq., \( \lambda = 3 \)

ODEs – numerical methods

- Euler’s method (= explicit Euler = Euler forward)

Very simple idea:

- Point 1 given (initial value): \( y(t_0) = y_0 \)
- Find approximative \( y(t_1) \): take step \( h \) along the tangent line to \( y(t_0, y_0) \) and plug in \( t_i \mapsto y_1 \)
- Now find tangent line to \( y_1, y_2, \ldots \)
- Repeat for \( y_2, y_3, \ldots \) until reaching the end of the interval

Note, \( y_k \) denotes the approximation of \( y(t_k) \)
and \( f_k \) denotes the approximative \( f(t_k, y_k) \)
The Euler method

**ODEs – numerical methods**

**The Euler method**

Tangent line in \( y(t_0) \)

Tangent line in \( f(t, y) \)
yields \( y_2 \)

Repeat until reaching the final point on interval

Mathematical formulation:

\[ y_{k+1} = y_k + h f(t_k, y_k) \]

Algorithm:

\[
\begin{cases}
  y_{k+1} = y_k + h f(t_k, y_k), & k = 0, 1, 2, \ldots, N \\
  y_0 = y(\cdot) 
\end{cases}
\]

Can be derived through Taylor expansion

**ODEs, errors**

**The Euler method**

What errors are introduced?

Error in step 1:

Error in step 2

Two kinds:

- **local error** – error introduced in this step (green arrow)
- **global error** – total error in this step (blue arrow)
**ODEs, errors**

*The Euler method*

Error in step 3
- local error — too small to show graphically here (grafiskt)
- global error — blue arrow

Error decreases as \( h \) getting smaller.

Discretization error depends in some way on \( h \) and can be derived using Taylor expansions, see ODE part 2 (later). Apart from discretization errors, there are roundoff errors, normally much smaller.

**ODEs — numerical methods**

*Runge-Kutta method (here Heun’s)*

Idea: Weigh several tangent lines together.

Example:
To find \( y_1 \), find tangent line in \( f_0 \) and \( f_1 \) and combine the two (the average).

Problem: How can one find tangent in when \( y_0 \) and \( f_1 \) not yet computed?

Answer: Use the Euler solution and move tangent line to point \( y_0 \).
ODEs – numerical methods

Runge-Kutta methods (here Heun’s)

... And weigh the two together (average in this case)

ODEs – numerical methods

Runge-Kutta methods (here Heun’s)

This yields

\[
\begin{align*}
y_{n+1} &= y_n + \frac{h}{2} \left[ f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n)) \right] \\
y_0 &= \hat{y}
\end{align*}
\]

Euler approximation

ODEs – numerical methods

Runge-Kutta methods

■ Classical R-K (a 4-stage R-K):

\[
\begin{align*}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\
k_4 &= f(t_n + h, y_n + h k_3) \\
y_{n+1} &= y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\
y_0 &= \hat{y}
\end{align*}
\]

ODEs – numerical methods

Runge-Kutta methods

■ General s-stage explicit R-K

\[
\begin{align*}
k_i &= f(t_j, y_j) \\
k_j &= f(t_n + \sum_{i=1}^{s-1} \frac{h}{a_i} k_i) \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i \\
y_0 &= \hat{y}
\end{align*}
\]

where \( a_{ij}, b_j \) and \( c_i = \sum_{j=1}^{s} a_{ij} \) are constants

ODEs – numerical methods

Runge-Kutta methods

■ Classical R-K (a 4-stage R-K):

\[
\begin{align*}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\
k_4 &= f(t_n + h, y_n + h k_3) \\
y_{n+1} &= y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\
y_0 &= \hat{y}
\end{align*}
\]

ODEs – numerical methods

Runge-Kutta methods

■ General s-stage explicit R-K

\[
\begin{align*}
k_i &= f(t_j, y_j) \\
k_j &= f(t_n + \sum_{i=1}^{s-1} \frac{h}{a_i} k_i) \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i \\
y_0 &= \hat{y}
\end{align*}
\]

where \( a_{ij}, b_j \) and \( c_i = \sum_{j=1}^{s} a_{ij} \) are constants

ODEs – numerical methods

Runge-Kutta methods

■ Classical R-K (a 4-stage R-K):

\[
\begin{align*}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\
k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\
k_4 &= f(t_n + h, y_n + h k_3) \\
y_{n+1} &= y_n + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \\
y_0 &= \hat{y}
\end{align*}
\]

ODEs – numerical methods

Runge-Kutta methods

■ General s-stage explicit R-K

\[
\begin{align*}
k_i &= f(t_j, y_j) \\
k_j &= f(t_n + \sum_{i=1}^{s-1} \frac{h}{a_i} k_i) \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i \\
y_0 &= \hat{y}
\end{align*}
\]

where \( a_{ij}, b_j \) and \( c_i = \sum_{j=1}^{s} a_{ij} \) are constants
ODEs – numerical methods

- How can you find the solution in point $k+1$ by using the solution in the same point?

- Example) $\frac{dy}{dt} = y^2$, \quad t \geq 0 \quad \text{i.e.} \quad f(t,y) = y^2$

  Implicit Euler, \quad h=0.1:\n  \begin{align*}
  y_{k+1} & = y_k + h f(t_k, y_k) \\
  \text{First step, } k=0: \quad & y_1 = y_0 + h f(t_0, y_0) = 1 + 0.1 \cdot y_0^2
  \end{align*}

  Non-linear equation – Use iterative method e.g. Newtons method.
  Let
  \[ y_1 - 1 + 0.1 \cdot y_1^2 = 0 \]

Ordinära diffar – metoder

- Example, cont.)
  An iterative solver (fzero in Matlab)
  \[ y_i = \text{iterativ}(g(y_i), y_{guess}) \]
  Possible start guess \[ y_{guess} = y_0 \]
  Repeat for every time-step

ODEs – numerical methods

- Another implicit method
  - Trapezoid method
    \[ y_{i+1} = y_i + 0.5h(f(t_i, y_i) + f(t_{i+1}, y_{i+1})) \]
  - It’s an implicit Runge-Kutta

- If a system of ODEs one a system of non-linear equations to solve every time step => makes these methods more work intensive

- But some properties these methods have, make them useful for special kinds of ODEs, see next part (stiff problems) – they might in total be much less work intensive