Mathematical Methods and Modeling
Laboratory class

Numerical Integration of Ordinary Differential Equations

Exact Solutions of ODEs

- Cauchy's Initial Value Problem in normal form:

$$\begin{cases} y'(x) = f(x, y(x)) & \text{for } x \in I = [x_0, x_F] \\ y(x_0) = y_0 \end{cases}$$

- Recall:
  - if f is locally Lipschitz-continuous
    → there is an unique local solution
  - if f is (uniformly) Lipschitz-continuous on all *I*
    → there is an unique solution in *I*, i.e. global solution

Numerical Analysis

- If solution can hardly be explicited → numerical

- Numerical Analysis is the branch of mathematics studying approximation methods for solving equations → applications on calculators

Explicit Euler Method

- For finding an approximate (or numerical) solution of an ODE in normal form, first discretize the domain *I* into *n* subintervals of width *h*:

$$x_0 < x_1 < x_2 < \ldots < x_n = x_F$$

s.t. $$x_{k+1} - x_k = h$$

- Discretize the derivative *y'(x)* with the (forward) finite difference:

$$y'(x_k) \simeq \frac{y(x_{k+1}) - y(x_k)}{h}$$

- Denote $y_k := y(x_k), \forall k = 0, \ldots, n$ and
- suppose to be able to calculate

$$f(x_k, y(x_k)) = f(x_k, y_k)$$

Explicit Euler Method

- The ODE can then be approximated as

$$y_{k+1} \simeq y_k + hf(x_k, y_k)$$

- Approximate solution:

$$\boxed{u_{k+1} = u_k + hf(x_k, u_k) \quad \text{for all } k = 0, \ldots, n-1}$$

- This formula for calculating the approximate solution is called Explicit/Forward Euler Method
- Necessary:
  - Initially: $x_0, u_0 = y_0 \in \mathbb{R}$
  - At each step: $f(x_k, u_k)$
  - Decide discretization step *h*
- It can be shown *consistent*: $h \to 0 \implies u \to y$

Implicit Euler Method

- Conversely, if we take the backward finite difference $$y'(x_k) \simeq \frac{y(x_k) - y(x_{k-1})}{h}$$

we have the approximation:

$$\boxed{u_{k+1} = u_k + hf(x_{k+1}, u_{k+1}) \quad \text{for all } k = 0, \ldots, n-1}$$

- This formula is called Implicit/Backward Euler Method
- However, in this method we have to solve the equation (with *f* appearing) in the unknown u_{k+1}
- In practice, when *f(x,y)* behaves badly, Implicit method is preferred if computationally feasible

Theta-Method

- Instead, fix θ in [0,1] and take the intermediate value of f:

$$y'(x_{k+1}) \simeq \theta f(x_{k+1}) + (1-\theta) f(x_k)$$

- Then it is easy to derive a generalization of EMs:

$$\boxed{u_{k+1} = u_k + h[\theta f(x_{k+1}, u_{k+1}) + (1-\theta) f(x_k, u_k)]}$$

- This formula is called the θ-method
- Clearly, with $\theta = 0, 1$ it is Explicit/Implicit EM
- Is it explicit or implicit?
- It has better convergence properties

Example

- Find the numerical solution to Cauchy's IVP

$$\begin{cases} y'(t) = 3t - ty(t) & \text{for } t \in I = [1,5] \\ y(1) = 1 \end{cases}$$

- Compare it with the exact solution $y(t) = 3 - 2e^{\frac{1}{2} - \frac{t^2}{2}}$

1. Define the M-file for function *f(t,y) = 3t-ty*
2. Fix *h>0* and find numerical solution *u* by EEM
3. Draw *y, u* and the error *y-u*

Example

- Let's try to numerically solve previous IVP with IEM. Unluckly, this time it is not possible to use MATLAB for inverting *f(t,y)* w.r.t. *y*. Hence, invert it with paper & pencil

$$\begin{aligned} z_{k+1} &= z_k + hf(t_{k+1}, z_{k+1}) = \\ &= z_k + h[3t_{k+1} - t_{k+1} z_{k+1}] \end{aligned}$$

- That yields

$$z_{k+1} = \frac{z_k + 3ht_{k+1}}{1 + ht_{k+1}}$$

- Compare IEM's to EEM's and exact solution *y*, finding *maximum absolute error* and *sum-of-squares error*

MATLAB ODE Solver

- Luckly, MATLAB already has algorithms for ODEs
- ode45 function finds approximate solutions for most of simple non-stiff problems. Basic syntax:

```
[T,Y] = ode45(@fun,[x0 xF],initvals);
```

  - @fun is handle to funcion fun defining *f(x,y)*
  - x0,xF are initial and final values of interval *I*
  - vector initvals contains Cauchy's initial value(s) of the solution(s): y(x0)
- odeset function can set some options of ODE Solver specified as additional argument. Eg.:

```
odeopt = odeset('RelTol',1e-4,'AbsTol',…
                [1e-4 1e-4 1e-5]);
```

Example

- Find the numerical solution to Cauchy's IVP

$$\begin{cases} y'(t) = 3t - ty(t) & \text{for } t \in I = [1,5] \\ y(1) = 1 \end{cases}$$

- Compare it with the exact solution $y(t) = 3 - 2e^{\frac{1}{2} - \frac{t^2}{2}}$

1. Define the function *f(t,y) = 3t-ty*
2. Call ode45 function
3. Draw *y, u* and the error *y-u*

Example

- Find the numerical solution to Cauchy's IVP

$$\begin{cases} y'(t) = t^3 y^2(t) - \frac{4y}{t} & \text{for } t \in I = [2, 10] \\ y(2) = -1 \end{cases}$$

- Compare it with the exact solution

$$y(t) = \frac{16}{t^4(-16\log(t) - 1 + 16\log(2))}$$

1. Define the function *f(t,y)*
2. Call ode45 function
3. Draw *y, u* and the error *y-u*

Higher order ODE

- A n-th order ODE $y^{(n)}=f(t,y,y',\ldots,y^{(n-1)})$ can be transformed into a system of n ODEs with n variables
- Example: equation of Van der Pol Oscillator

$$x'' - \mu(1-x^2)x' + x = 0$$

after assigning $y=x'$, it can be rewritten as system

$$\begin{cases} x' = y \\ y' = \mu(1-x^2)y - x \end{cases}$$

- Try solving through MATLAB with

$$\mu = 2, x(0) = 2, y(0) = 0$$

in the interval [0, 20]

Higher order ODE

- Create the M-file vaderpol.m defining the vector function *f(t,y)* needed by ode45:

```
function out = vanderpol(t,x)
    %as stated in the documentation this function has
    %to take as arguments: time t and state-variable x.
    mu = 2; %parameter of the Van der Pol oscillator

    out = [0; 0];
    out(1) = x(2);
    out(2) = mu*(1-x(1)^2)*x(2) - x(1);
end
```

Higher order ODE

- Then in the command window run the instruction

```
[T,Y] = ode45(@vanderpol, [0 20], [2 0]);
```

- and draw the trajectory:

```
plot3(T,Y(:,1),Y(:,2));
xlabel('t'); ylabel('x'), zlabel('y');
```