Introduction

This training unit is focused on analysing and implementing solvers. They are common to all simulation software, because simulating physical systems implies solving ordinary differential equations and this is done by numerical methods that iteratively approximate the solution step-by-step. Those numerical methods are the so-called solvers –odeXX in the MATLAB's context.

The basis for understanding solvers we have already analysed in our previous lecture, but "nobody gets drunk by knowing the definition of wine!". Thus, in order to get a deeper understanding of the operation and importance of solvers, before using the solvers provided by Matlab and Simulink, we are going to program a simple, but representative solver. This "home-made" solver will be enhanced and covers the crucial aspect of time-variable solvers: handling of step size! The latter is strongly binding to the accuracy of results. You will require your programing skills for this exercise though.

Preparation

Read and analyze your notes of the solver lecture

Laboratory work

1.1 Warming-up

Let's start creating a basic solver

Create the algorithm for implementing the Forward Euler (FE) Method for solving Ordinary Differential Equation (ode), which we analyzed on the previous lecture

$$y_{k+1} = y_k + h \cdot f(t_k, y_k)$$

Take your time to understand how the Forward Euler (FE) works

a) Explain how this method approach (approximate) the objective function?

Hint: For a first analysis it is also possible to implement the FE-Method in Excel

b) Create a MATLAB script called "test_ode1.m" and integrate the FE-Method. Please make sure that you use a directory in which you have write permission!

c) Create a MATLAB function called "f.m". This function shall contain the target original differential equation $f(y,t)$,, stated in Table 1, for the start consider the EQ1

d) Plot the gathered result, use proper labels, title and legend

Table 1: Table of ODE's to be analysed

Number	$\frac{dy}{dt}$	ODE Exact solution y(t)	Parameters
EQ1	$\frac{y}{\tau}$	$y \cdot e^{-t/\tau}$	$y(0) = 1; \tau = -0.5;$
EQ2	t	$1 + t^2/2$	
EQ3	y	e^t	
EQ4	$-y$	e^{-t}	
EQ5	$\frac{1}{1-3t}$	$1 - \frac{\log(1-3t)}{3}$	
EQ6	$2y - y^2$	$\frac{2}{1+e^{-2t}}$	

2 Analysis on solver precision

Now, you have understood the background information –then let's start building over this!

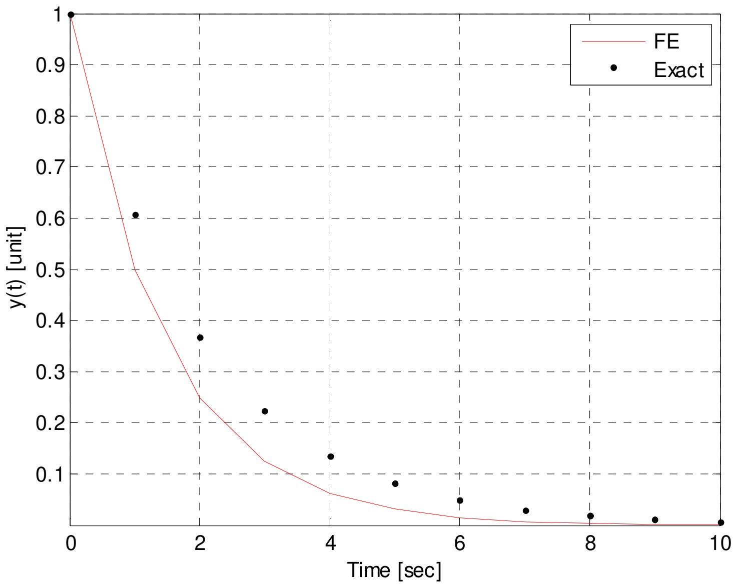

Fig. 1 – Comparison FE solution (solid line) and exact solution (dotted line)

a) Use "test_ode1.m"

Add the iterative computation of the exact solution of the differential equation, which is for e.g.:

$$\text{EQ1:} \quad y(t) = y_0 \cdot e^{-t/\tau}$$

Suggestion for handling: Create a matrix named "yk_ex" which groups the solution and each step together with the time t and the step size h

b) Combine the result of the FE method with the exact solution within a single plot

Note: You should get a plot similar to Fig. 1

c) Calculate the local error, as the difference at each step, between FE and exact solutions, add it to the plot

Quantify the average absolute total error

d) Test your implemented FE solver for solving the different equations (EQ1 - EQ6) of table1

What do you observe? Plot the results and make a statement

2.1 Analysis of step size

a) Try out different values of *h*: 0.1, 0.5, 1.5, 2, 3, 5

What do you observe? How does a change of *h* affect the result?

Which value is a "good" value for *h*? Why?

b) At what value of *h* does the answer oscillate? And at which value it becomes unstable?

c) Can you deduce any correlation between the selected value of *h* and τ of *EQ1*? If yes, report your finding and confirm it by trying a different value of Tau.

3 Creating a general-purpose solver

As you know, in the common practice the **exact solution of the differential equation is unknown**; therefore, the way to obtain an accurate result is by applying a second algorithm and comparing both results against an acceptance criterion. Thus, let's proceed as follows.

3.1 Runge-Kutta second order method (Midpoint)

$$k_1 = f(t_k, y_k)$$

$$k_2 = f\left(t_k + \frac{h}{2}, y_k + k_1 \cdot \frac{h}{2}\right)$$

$$y_{k+1} = y_k + h \cdot k_2$$

Note: Use an additional set of y-variables, e.g. yk_mp

Take your time to understand how the Midpoint (MP) Method works

a) Explain how this method approach (approximate) the objective function?

b) Rename your previous script as "test_ode12.m" and implement the Mid-Point (MP) method

c) Plot the results of the MP method together with FE and exact solution (Consider proper labels etc.)

d) Try out different values of h
   What do you observe?
   Which method is better? Why?

3.2 Add the variable step size calculation

e) Add the relative tolerance parameter *rtol* to your script with a value of 0.1 for the starting

f) You need to integrate the calculation of the relative error, after calculating the actual values of both methods

$$\varepsilon = \left| y_{k_{MP}} - y_{k_{FE}} \right|$$

g) If ε <= *rtol* then the actual step size can be kept; else recalculate the step size *h* (see next formula) and go for a next iteration

$$h = h \cdot sqrt\left(\frac{rtol}{\varepsilon}\right)$$

h) Think on protecting the algorithm making to small time steps and give a warning if occurs

i) Add representative plot and display the results show the variation of the step size

j) Test your variable step solver for solving the different equations (EQ1 - EQ6) of table1
   Analyse the influence of *rtol* on *h* by modifying *rtol*. Try out different values and explain the effect!

Hint: If something goes wrong and your program hangs-on then make active the command window and push Ctrl + C

4 Additional references

For particular information about use of solvers in MATLAB/Simulink, I strongly recommend to look at the following links:

https://www.mathworks.com/help/simulink/ug/types-of-solvers.html

http://www.mathworks.de/company/newsletters/articles/stiff-differential-equations.html

https://de.mathworks.com/moler/chapters.html