MBD: Training Unit 1 SOLVERS

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 cover 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; you are not going to start from scratch, since you are going to build your solver step-by-step upon the basis of a given example.

Preparation

Read and analyze your notes and presentation of the previous lecture

Laboratory work

1. Warming-up

Let's start using a given example

1. Download to your working directory the programs "test_ode1.m" and "f.m" from Moodle. Please make sure to use a directory where you have write permission!

2. Open and run the script "test_ode1.m"

   Note 1: This script implements the Forward Euler (FE) Method for solving Ordinary Differential Equation (ode), which we analyzed on the previous lecture:
   $$y_{k+1} = y_k + h \cdot f(t, y)$$

   Note 2: The target differential equation, contained in "f.m", is:
   $$\frac{dy}{dt} = -\frac{y}{\tau}$$

3. Take the time to understand how this program works
   a. Which are the main parameters?

b. How is the target function evaluated?

2. Analysis on solver precision

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

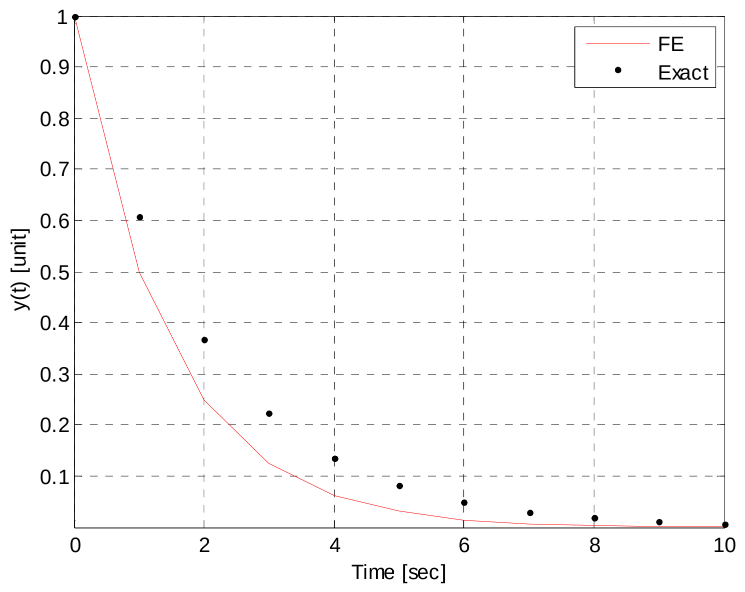

Fig. 1 – Comparison FE solution (solid line) and exact solution (dotted line)

4. Use "test_ode1.m"
   a. Add within the while-loop the iterative computation of the exact solution of the differential equation, which is:

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

   Suggestion: Generate a matrix called "yk_ex" which groups the solution and every step together with the time *t* and step size *h*

   b. Display and label the FE method result together with the exact solution
      Note: You should get a plot like Fig. 1
   c. Include the calculation of the local error as the difference at every step between FE and exact solutions
   d. Quantify the average absolute total error

5. Try out different values of *h*: 0.1, 0.5, 1.5, 2, 3, 5
   a. What do you observe? What is the influence of *h*?
   b. Which is a "good" value for *h*? Why?
   c. At what value of *h* does oscillate the response? And at what does it get instable?
   d. Can you deduce any correlation between the selected value of *h* and *Tau*? If yes, report your finding and confirm it by trying a different value of Tau.

3. Creating a general purpose solver

6. 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:
    a. Rename your previous script as "test_ode12.m" and implement the Mid-Point (MP) method, which is a particular case of Runge-Kutta second order method:

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

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

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

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

    b. Plot the results of the MP method together with FE and exact solution
    c. Try out different values of *h*. What do you observe? Which method is better? Why?

7. Add the step size calculation
    a. Add the relative tolerance parameter *rtol* to your script with a value of 0.1
    b. Within the exist while-loop you need to integrate an additional one that after calculating the actual values of both methods:
        i. Calculate relative error

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

        ii. If ε <= *rtol* then break the while-loop; else recalculate the step size *h* (see next formulas) and go for a next iteration

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

8. Add representative plot and display of results

9. Test your solver and analysis the influence of *rtol* on *h. Try out different values!*
   Hint: If something goes wrong and your program hangs-on then make active the command window and push Ctrl + C

10. If you have time and curiosity! Here there are some additional differential equations you can solve with your program. In case you make it, include the results in your report

f(t,y)	Exact solution y(t)
t	1+t^2/2

y	exp(t)
-y	exp(-t)
1/(1-3*t)	1-log(1-3*t)/3 (Singular)
2*y-y^2	2/(1+exp(-2*t))

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