Warsaw, 30.05.2019

Assignment C: Solving ordinary differential equations

Final Report

Course: ENUME

Author: Kacper Kamieniarz

Index number: 293065

Advisor: Ph.D. Andrzej Miękina

Table of Contents

Task 1

Overview

First task of this assignment introduces a second order differential equation which will be the subject of all following tasks.

The equation is:

$$9y'' + 6y' + 10y = 0$$

with domain $t \in [0, 10]$ and initial conditions $y(0) = 0\ and\ y'(0) = 2$.

The goal of this task is to develop a program for solving the above differential equation by means of the implicit Gauss-Legendre order 6 method defined by the following Butcher table:

$$\begin{array}{c|ccc}
\frac{1}{2}-\frac{\sqrt{15}}{10} & \frac{5}{36} & \frac{2}{9}-\frac{\sqrt{15}}{15} & \frac{5}{36}-\frac{\sqrt{15}}{30} \\
\frac{1}{2} & \frac{5}{36}+\frac{\sqrt{15}}{24} & \frac{2}{9} & \frac{5}{36}-\frac{\sqrt{15}}{24} \\
\frac{1}{2}+\frac{\sqrt{15}}{10} & \frac{5}{36}+\frac{\sqrt{15}}{30} & \frac{2}{9}+\frac{\sqrt{15}}{15} & \frac{5}{36} \\
\hline
 & \frac{5}{18} & \frac{4}{9} & \frac{5}{18}
\end{array}$$

The solution obtained in this way shall be compared with a solution obtained by means of MATLAB operator `ode113`.

Functions

- **`dydt = f(t, y)`**

Input	t – vector of points at which f is evaluated, y – matrix of values of y_1 and y_2
Output	dydt – matrix of derivatives of of y_1 and y_2

Function that makes a substitution in the second order differential equation and returns a vector of first order derivatives of y.
The substitution: $y_2 = y_1'$ yields a system of equations

$$\begin{cases} y_1' = y_2 \\ y_2' = -\frac{10}{9}y_1 - \frac{2}{3}y_2 \end{cases}$$

so the returned vector has the form

$$y = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2 \\ -\frac{10}{9}y_1 - \frac{2}{3}y_2 \end{bmatrix}$$

- **`[x, u] = myODE(tspan)`**

Input	tspan – interval of t for solving ODE
Output	x – tspan, u – matrix of results of solution

Function for solving the above second order differential equation using 6th order Runge-Kutta implicit method. It takes tspan as argument which is a vector of t values at which the function is evaluated.
Using general formula for one-step Runge-Kutta methods the following equations were obtained.

$$y_n = y_{n-1} + h(\frac{5}{18}k_1 + \frac{4}{9}k_2 + \frac{5}{18}k_3)$$

with:

$$k_1 = f(t_{n-1} + \left(\frac{1}{2} - \frac{\sqrt{15}}{10}\right)h, y_{n-1} + h\left[\frac{5}{36}k_1 + \left(\frac{2}{9} - \frac{\sqrt{15}}{15}\right)k_2 + \left(\frac{5}{36} - \frac{\sqrt{15}}{30}\right)k_3\right])$$

$$k_2 = f(t_{n-1} + \frac{1}{2}h, y_{n-1} + h\left[\left(\frac{5}{36} + \frac{\sqrt{15}}{24}\right)k_1 + \frac{2}{9}k_2 + \left(\frac{5}{36} - \frac{\sqrt{15}}{24}\right)k_3\right])$$

$$k_3 = f(t_{n-1} + \left(\frac{1}{2} + \frac{\sqrt{15}}{10}\right)h, y_{n-1} + h\left[\left(\frac{5}{36} + \frac{\sqrt{15}}{30}\right)k_1 + \left(\frac{2}{9} + \frac{\sqrt{15}}{15}\right)k_2 + \frac{5}{36}k_3\right])$$

then:

$$k_1 = A \cdot [y_{n-1} + h\left[\frac{5}{36}k_1 + \left(\frac{2}{9} - \frac{\sqrt{15}}{15}\right)k_2 + \left(\frac{5}{36} - \frac{\sqrt{15}}{30}\right)k_3\right]]$$

$$k_2 = A \cdot [y_{n-1} + h\left[\left(\frac{5}{36} + \frac{\sqrt{15}}{24}\right)k_1 + \frac{2}{9}k_2 + \left(\frac{5}{36} - \frac{\sqrt{15}}{24}\right)k_3\right]]$$

$$k_3 = A \cdot [y_{n-1} + h\left[\left(\frac{5}{36} + \frac{\sqrt{15}}{30}\right)k_1 + \left(\frac{2}{9} + \frac{\sqrt{15}}{15}\right)k_2 + \frac{5}{36}k_3\right]]$$

The above yields a matrix system of equations:

$$\begin{bmatrix} I-\frac{5}{36}hA & \left(-\frac{2}{9}+\frac{\sqrt{15}}{15}\right)hA & \left(-\frac{5}{36}+\frac{\sqrt{15}}{30}\right)hA \\ \left(-\frac{5}{36}-\frac{\sqrt{15}}{24}\right)hA & I-\frac{2}{9}hA & \left(-\frac{5}{36}+\frac{\sqrt{15}}{24}\right)hA \\ \left(-\frac{5}{36}-\frac{\sqrt{15}}{30}\right)hA & \left(-\frac{2}{9}-\frac{\sqrt{15}}{15}\right)hA & I-\frac{5}{36}hA \end{bmatrix} \cdot \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix} = \begin{bmatrix} A\cdot y_{n-1} \\ A\cdot y_{n-1} \\ A\cdot y_{n-1} \end{bmatrix}$$

Results

Using constant integration step $h = 0.01$ by passing `tspan = linspace(0, 10, 1001);` a solution was obtained that is further presented on figure 1.1. For comparison the ODE has also been solved using `ode113` operator. An options structure for ODE solver has been created. The structure sets the absolute tolerance AbsTol value to 10^{-16} and relative tolerance RelTol parameter to 10^{-16}. Figure 1.2. presents graph of dependency y_1 and y_2 on t.

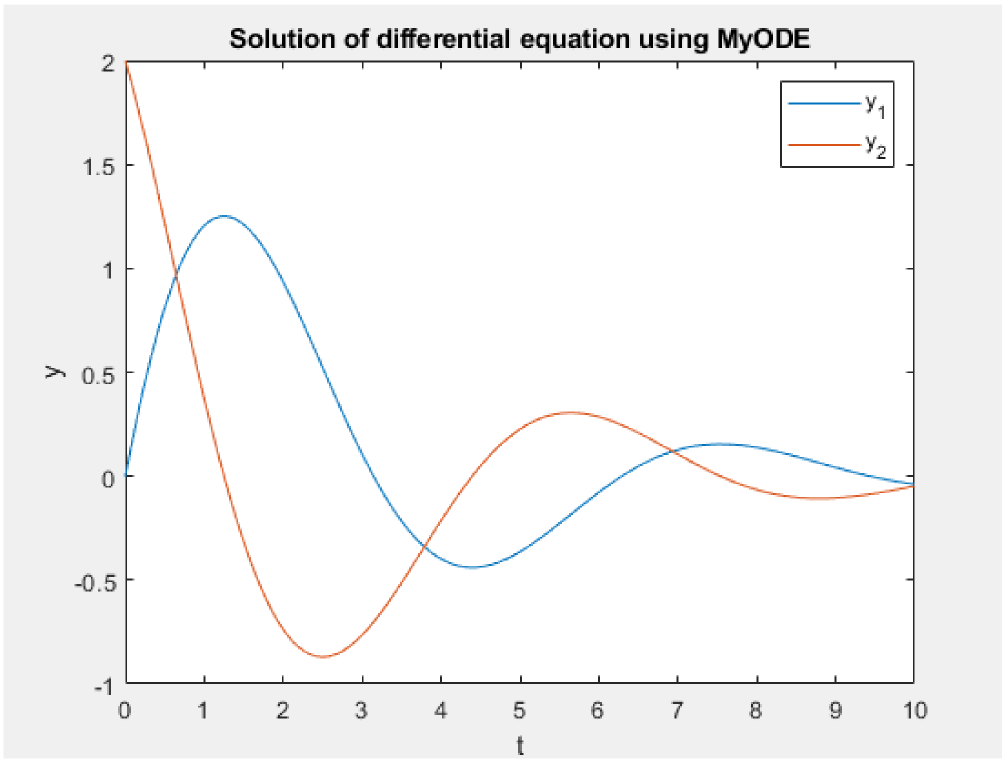

Figure 1.1

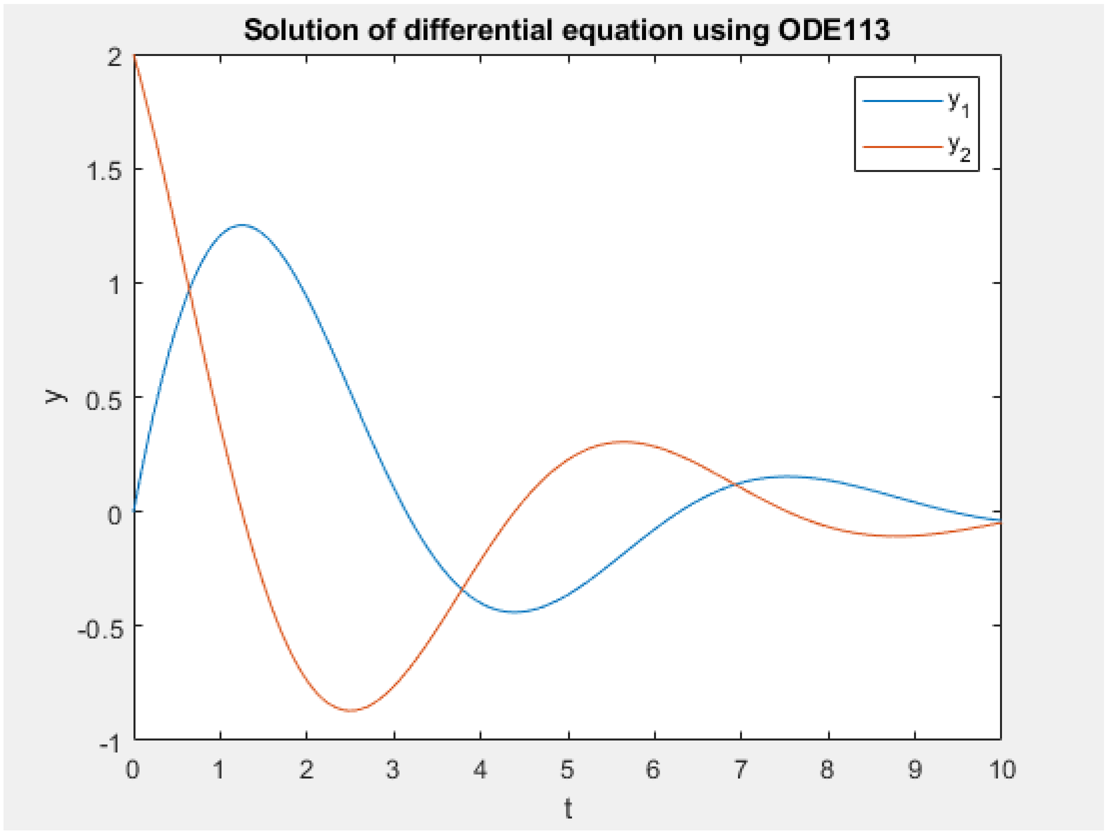

Figure 1.2

Conclusions

Comparing two graphs obtained by two methods shows that the method used in `myODE` function with step $h = 0.01$ yields an almost identical approximation of the solution, any differences are not visibly seen on the attached graphs. However manipulating the integration step value revealed that the bigger it is, the less accurate the solution of the equation is. Passing a vector with 10001 points t does not make a big difference to the `ode113` solver because either way when we plot solutions obtained with `myODE` and `ode 113` on one figure, they blend into each other and are indistinguishable.

Task 2

Overview

This part of the assignment focuses on investigating the dependence of the accuracy of the solution with `myODE` method on the integration step h. For that purpose the following indicators are used.

$$\delta_2(h) = \frac{\left\| \hat{\mathbf{y}}(t;h) - \dot{\mathbf{y}}(t,h) \right\|_2}{\left\| \dot{\mathbf{y}}(t,h) \right\|_2} \quad \text{(the root-mean-square error)}$$

$$\delta_\infty(h) = \frac{\left\| \hat{\mathbf{y}}(t;h) - \dot{\mathbf{y}}(t,h) \right\|_\infty}{\left\| \dot{\mathbf{y}}(t,h) \right\|_\infty} \quad \text{(the maximum error)}$$

Functions

- **`error = RMS(h, func)`**

Input	h – the integration step, func – function of which accuracy is tested (in this task `myODE`)
Output	error – vector of error values

Function for computing the root-mean-square error following the formula given in the task.

- **`error = ME(h, func)`**

Input	h – the integration step, func – function of which accuracy is tested (in this task `myODE`)
Output	error – vector of error values

Function for computing the maximum error following the formula given in the task.
Both of the above functions produce a vector of t values:

```
span = linspace(0, 10, 10/h +1);
```

which is then passed both to `ode113` and `myODE` which create matrices of solutions for given span. The error between those solutions is then returned by the functions.

Results

For the purpose of investigating the accuracy, a vector h is created using the function `h = logspace(-5, -1, 20)` which generates a row vector h of 20 logarithmically spaced points between decades 10^{-4} and 10^{-1}.
For each value from the vector h the root-mean-square as well as maximum errors are computed using functions `RMS(h, @myODE)` and `ME(h, @myODE)`.

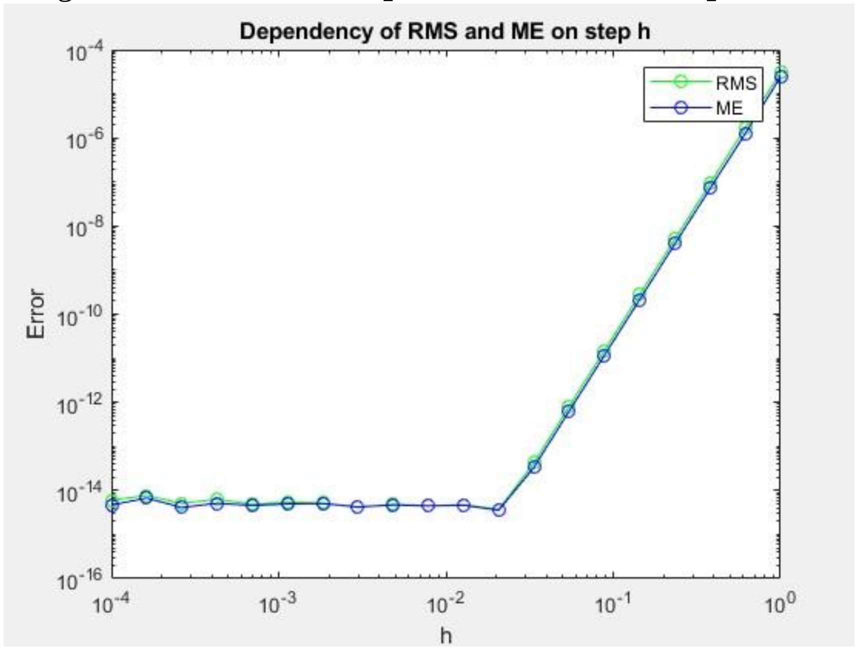

Figure 2.1

Conclusions

Figure 2.1. presents the root-mean-square and the maximum error dependence on the step value h. Both axes of this figure are scaled logarithmically. For integration step values between 10^{-4} and 10^{-2} the error values sit around 10^{-14} but approaching 10^{-1} they start to grow rapidly. The maximum error values are slightly smaller than root-mean-square but are very close to each other. Those figures show that the Runge-Kutta order 6 method used in this task loses its precision for integration step values greater than approximately $3.5 \cdot 10^{-15}$.

Task 3

Overview

This task involves solving the second order differential equation from Task 1 using the forward Euler's explicit method and comparing solution obtained by means of this method with Runge-Kutta order 6 method from Task1. For that purpose the root-mean-square error and the maximum error operators as defined in Task 2 are used.

Functions

- **`[x, u] = euler(tspan)`**

Input	tspan – interval of t for solving ODE
Output	x – tspan, u – matrix of results of solution

Function for solving the ODE using forward Euler's explicit method given by formula

$$y_n = y_{n-1} + h \cdot f(t_{n-1}, y_{n-1})$$

It solves the equation

$$9y'' + 6y' + 10y = 0$$

with initial conditions $y(0) = 0 \; and \; y'(0) = 2$ for given `tspan`.

- **`error = RMS(h, func)`**

- **`error = ME(h, func)`**

Same functions as in Task 2 are used but this time, the argument `func` passed to them is `@euler` function designed for solving the ODE using forward explicit Euler's method.

Results

Similarly as in Task 2 a vector of 20 logarithmically spaced points between 10^{-5} and 10^{-6} is created using `logspace()` operator. Root-mean-square and Maximum error vectors are created using functions `RMS(h, @euler)` and `ME(h, @euler)`.
Figure 3.1. presents the dependence of the root-mean-square error on the integration step h for both implicit Runge-Kutta order 6 method from Task 1 and forward Euler's explicit introduced above.
Figure 3.2.analogicaly presents the maximum error dependence on the integration step h for these methods.

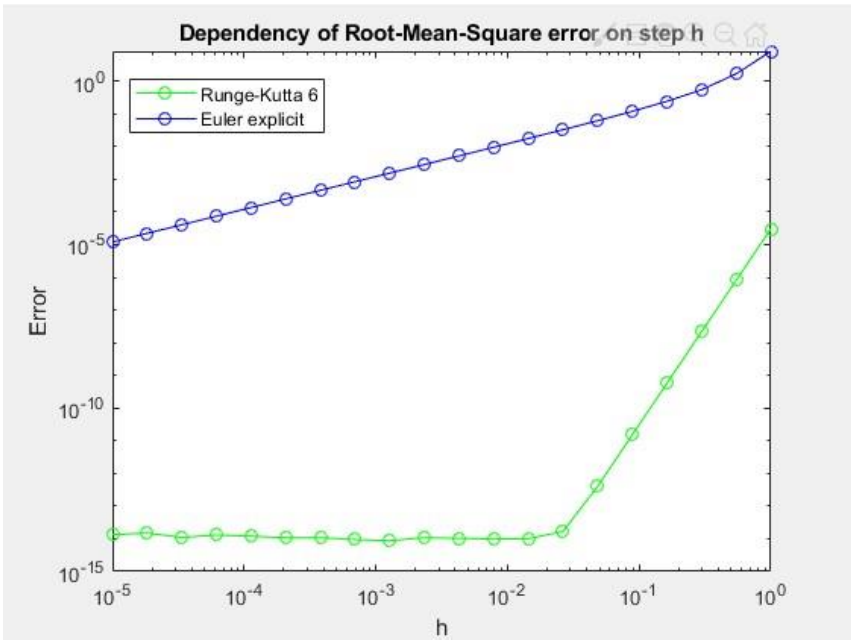

Figure 3.1

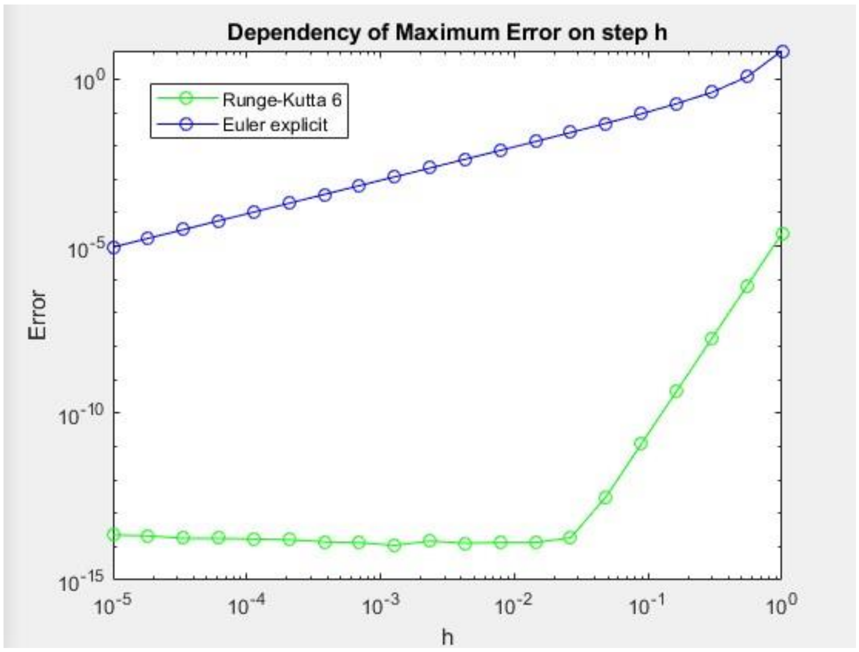

Figure 3.2

Conclusions

The root-mean-square error for forward Euler's explicit method increases almost linearly with increase of the integration step h. The slope slightly increases between decades 10^{-1} and 10^{0}. Unlike in Runge-Kutta case, the increase of error within increase of step h shows on the entire interval.

The reason for which the errors are greater in case of the forward Euler's explicit method is that the Euler's method is a first order Runge-Kutta, whereas the Gauss-Legendre is an order 6 method, therefore single step local error is greater by definition for Euler's method and it is given by

$$r_n(h) \cong -\frac{1}{2}\dot{y}_n'' h^2$$

where

$$y(t_n) \equiv \dot{y}_n$$

Moreover looking at the figures for the errors one can conclude that in case of the forward Euler's explicit method, the value of error in both cases (RMS and ME) is very close to the value of the integration step h.

References

- https://www.mathworks.com/help/matlab/
- Roman Z. Morawski – Numerical Methods (ENUME) Lecture notes for Spring Semester 2018/2019