Technical University of Denmark

02685 Scientific Computing for Differential Equations
2017

Assignment 1

Authors:
Miguel Suau de Castro (s161333)
Michal Baumgartner (s161636)
March 16, 2017

Contents

1 The Test Problem and DOPRI54

In this first section we are going to implement a set of numerical methods for solving ordinary differential equations. Since the algorithms are only approximations to the real solution, we shall also test their accuracy and discuss their performance by comparing the results obtained when solving the two following initial value problems:

$$\dot{x} = \lambda x(t) \qquad x(0) = 1 \qquad \lambda = -1 \tag{1}$$

$$\ddot{x} = -x(t) \qquad x(0) = 1 \qquad \dot{x(0)} = 0 \tag{2}$$

1.1 One step methods

As a first approach, we will apply Explicit Euler's method. This basic algorithm makes use of finite difference methods to replace the derivatives in the differential equation.

Considering a differential equation of the form:

$$\frac{dy}{dt} = f(t_n, y_n) \qquad y(t_0) = y_0 \tag{3}$$

If the independent variable is discretized, the solution can be obtained in n steps, by performing cosequtive approximations to the real function values.

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{4}$$

Where h is then the size of the step. It is easy to show that the smaller the step size is the more accurate our solution will be.

Instead of using the previous iterate one could also look at future values to approximate a solution. This method is called backward or implicit Euler:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{5}$$

However, for nonlinear problems the solution of the previous equation may require the use of numerical solvers, such as Newton's method, and thus the algorithm becomes computationally more demanding than the explicit Euler's method. We shall see in the next section the advantange of using this method.

Finally, the trapezodial method can be seen as a combination of both methods, where the solution is computed in every iteration by taking the average of the forward and backward finite difference approximations.

$$y_{n+1} = y_n + \frac{1}{2}h(f(t_{n+1}, y_{n+1}) \tag{6}$$

As in the previous case, 6 is an implicit equation and thus it might require the use of Newton's method to find a solution.

Figure 1 shows the solution from $t = 0$ to $t = 10$ of the two initial value problems given by explicit, implicit Euler and trapezoidal, along with the true solution. The solution is approximated in both cases by using 30 points ($h = 0.345$).

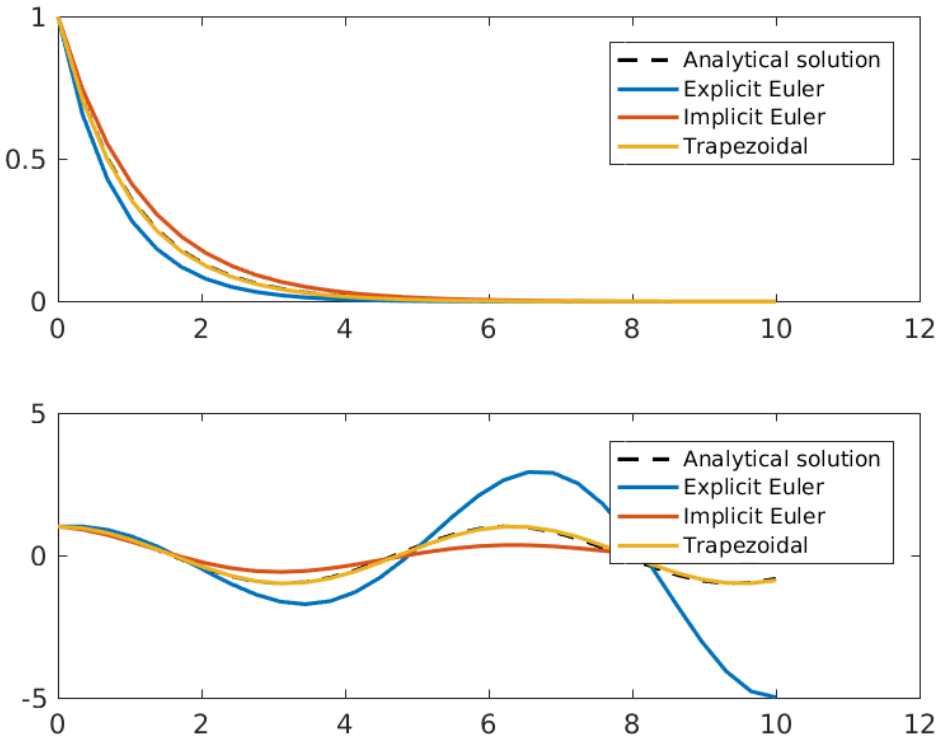

Figure 1: Explicit, Implicit Euler and Trapezoidal method solution to test equation and harmonic oscillator for h = 0.345

All three implimentations of these one step algorithms along with the code for Newton's method can be found in the appendix.

1.2 Multistep methods

Even though we see in figure 1 that the solution found by the trapezoidal method lies very close to the analytical solution, for some applications one could require higher order approximations to obtain more accurate results.

The classical Runge-Kutta method and its higher order variations play with the concept introduced in last section to create finer approximations to the solution. Concretetly the classical Runge-Kutta method takes four stages to compute a weighted avarage of four different slope estimates. The slope estimation at the fourth stage will be based in the information obtained in the three previous stages.

$$
\begin{aligned}
T_1 &= t_n & X_1 &= x_n \\
T_2 &= t_n + \frac{1}{2}h & X_2 &= x_n + \frac{1}{2}hf(T \\
T_3 &= t_n + \frac{1}{2}h & X_3 &= x_n + \frac{1}{2}hf(T \\
T_4 &= t_n + h & X_4 &= x_n + hf(T_3, \\
t_{n+1} &= t_n + h & & \\
x_{n+1} &= x_n + h(\frac{1}{6}f(T_1, X_1) + \frac{1}{2}f(T_2, X_2) + \frac{1}{2}f(T_3, X_3) + \frac{1}{6}f(T_4, X_4)) & &
\end{aligned}
\tag{7}
$$

Equation 7 reveals that since this is an explicit method there is no need for a numerical solver. The coefficients and weights of these equations are often collected in the Butcher's Tableu. This is described more deeply in section 3.

Two different variations of DOPRI54 have also been implemented. DOPRI54 is just another explicit Runge-Kutta method of higher order with more stages and a larger Butcher's Tableu. We shall see when comparing the error estimates the difference between the two methods. The three methods are shown in the appendix.

1.3 Global and local errors

It is easy to see in figure 1, especially in the bottom graph, that, since we base the solution at one point on previous approximations, the further the points are from the initial value the more inaccurate they become and the greater the distance to the true solution is. This distance is called global error, whilst the error made in every iteration is known as local error. As we will discuss later the latter is commonly used to classify different methods depending in their accuracy.

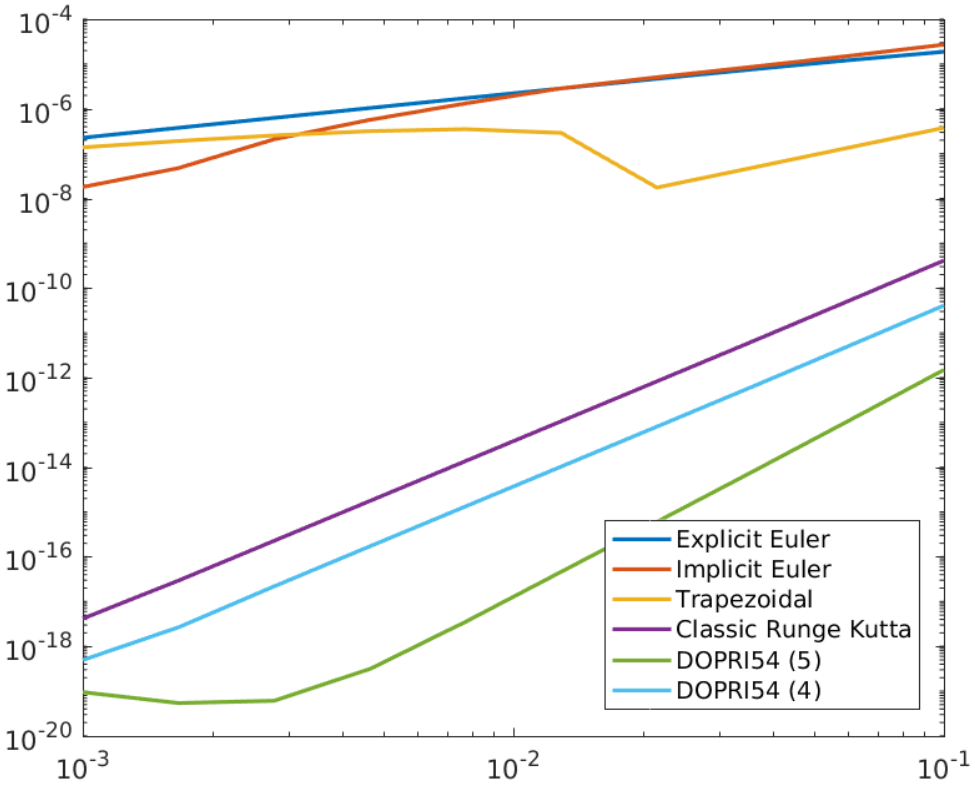

Figure 2: Explicit, Implicit Euler and Trapezoidal method global errors in test equation

One could then derive the analytical expression of the solution for both problems and compute te local and global errors.

$$x(t) = \exp(-t) \qquad x(t) = cos(t) \tag{8}$$

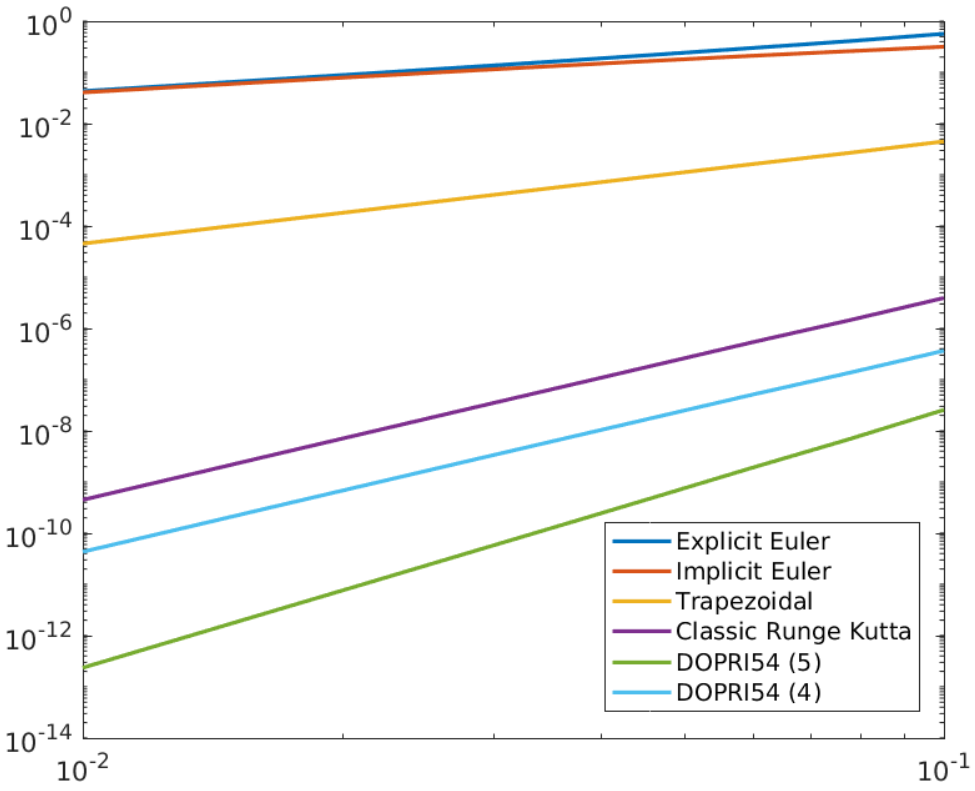

Figure 3: Explicit, Implicit Euler and Trapezoidal method global errors in harmonic oscillator

Figure 2 represents the global error at time $t = 10$ made by the one step solvers (left graph) and the multistep solver (right graph) for different step sizes. As expected, the size the global error decreases when increasing the number of points used in the approximation.

Why is not good to use global error??

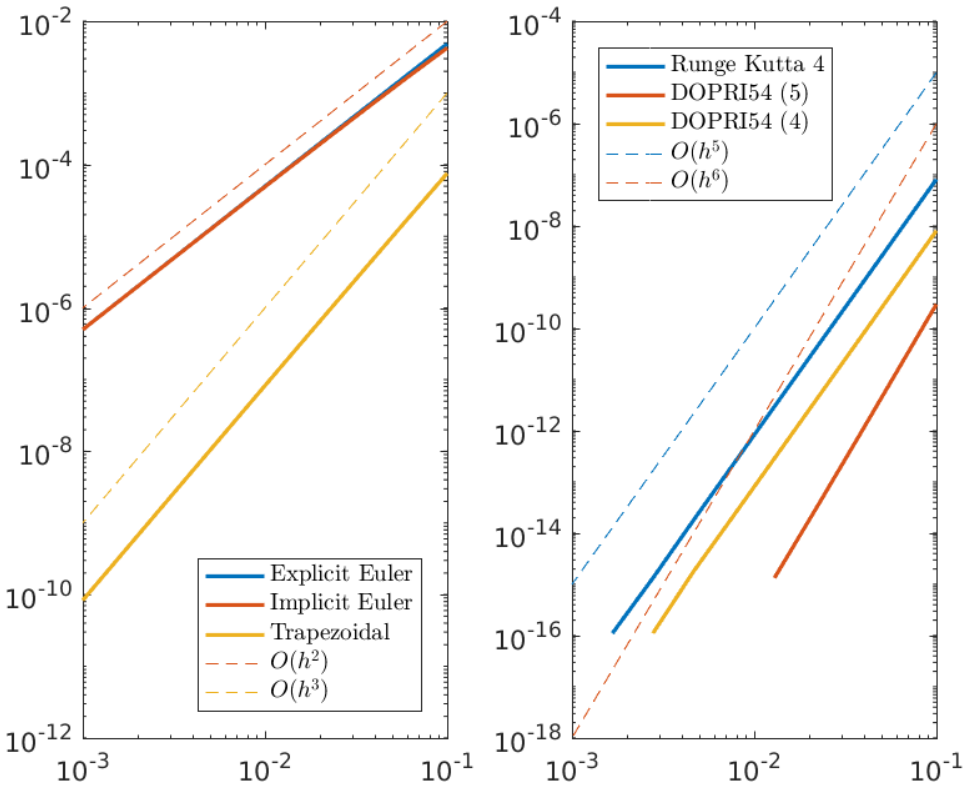

Figure 4: Explicit, Implicit Euler and Trapezoidal method local errors in test equation

On the other hand the local error at time $t = t_0 + h$ is shown in figure 4. Again we see how the error decreases with the step size. Moreover, as the plots use logarithmic scale and the curves are approximately straight lines, we can conclude that there is an exponential dependence between local error and step size or in big O notation: $O(h^{p+1})$. The constant p is used to characterize different methods, thus we say that a method is order 2 when the local error is proportional toh^3. The dashed lines in figure **??** can be used to verify the order of the solvers. That is, order 1 for Explicit and Implicit Euler, order 2 for the Trapezoidal method, order 4 for the classical Runge Kutta and one of the DOPRI54 versions and order 5 for the other DOPRI54.

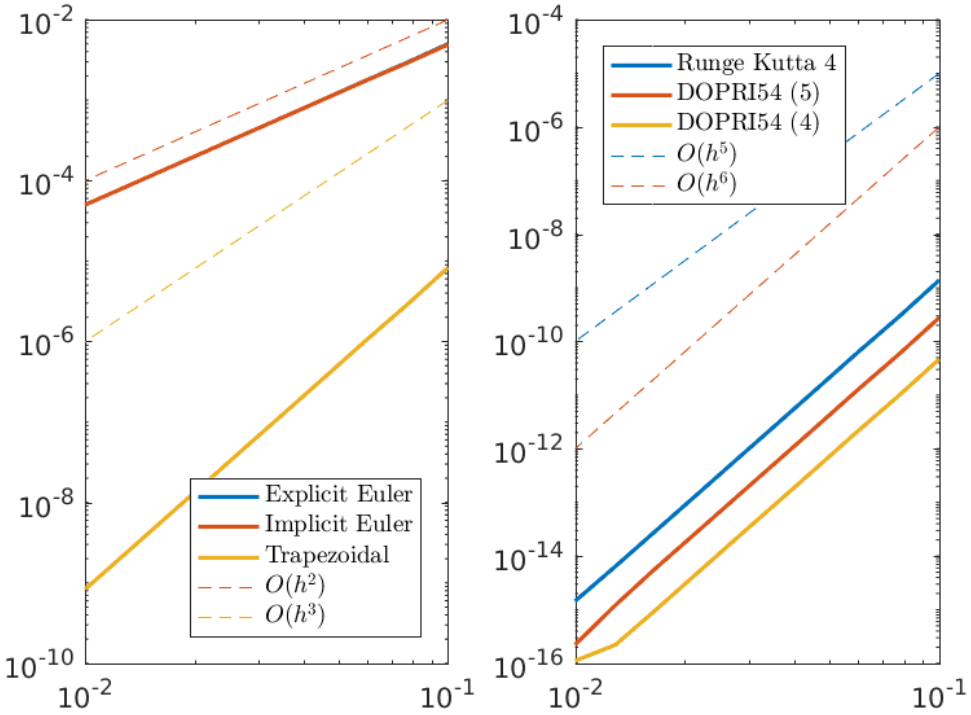

Figure 5: Explicit, Implicit Euler and Trapezoidal method local errors in harmonic oscillator

1.4 Error estimation

Considering that the algorithms are used to solve differential equations that are hard to derive analyticaly, calculate the exact error is not always possible.

An easy way to estimate the local error is called step doubling. The solution is computed for ... (performance). It turns out that estimate is proportional to the exact error and we can then

More sophisticated algorithms, such as DOPRI54, use embbeded methods of lower order to estimate the error. Even though the estimations obtained as not as good as with step doubling, the secondary method is closely related to the main algorithm so that they can share computations and thus be very efficient.

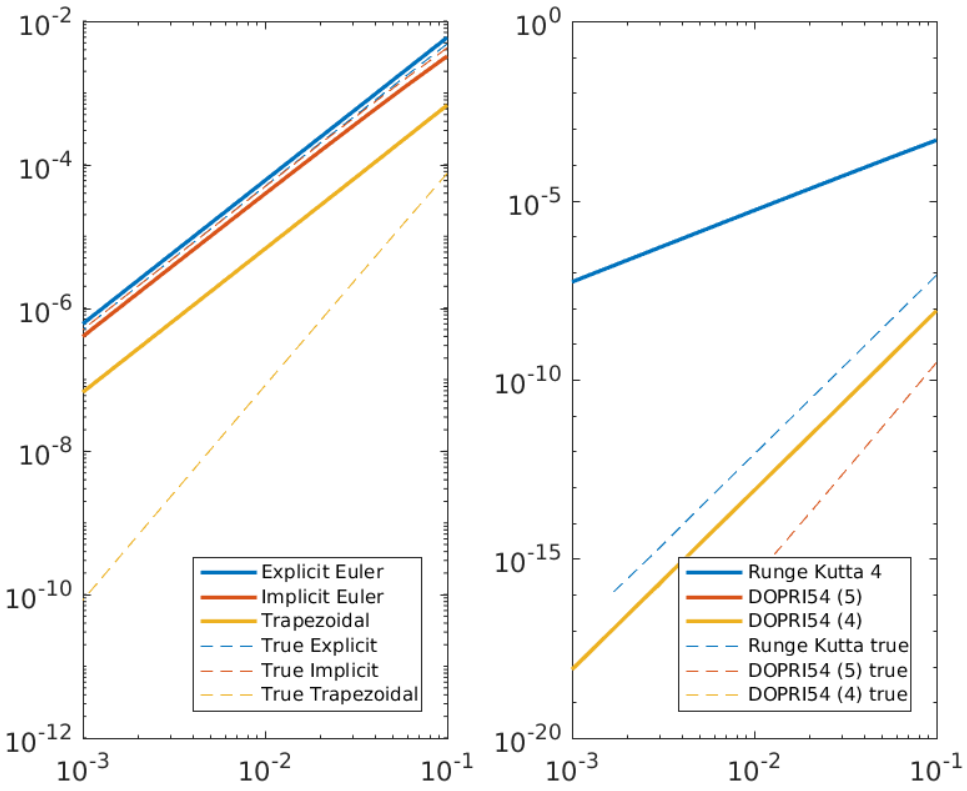

Figure 6: Explicit, Implicit Euler and Trapezoidal method local error estimates in test equation

The local error estimates are plotted along with the true errors in figure 6 for different step sizes. Even though, they do not match the exact values for some of the methods, the estimates lie always above the true errors which means that they can be used as an upper bound. Besides, as we know that the estimates are proportional to the exact local errors their slope can be used to verify the order of accuracy. In the embedded method estimation the slope is related to the order of lower order algorithm.

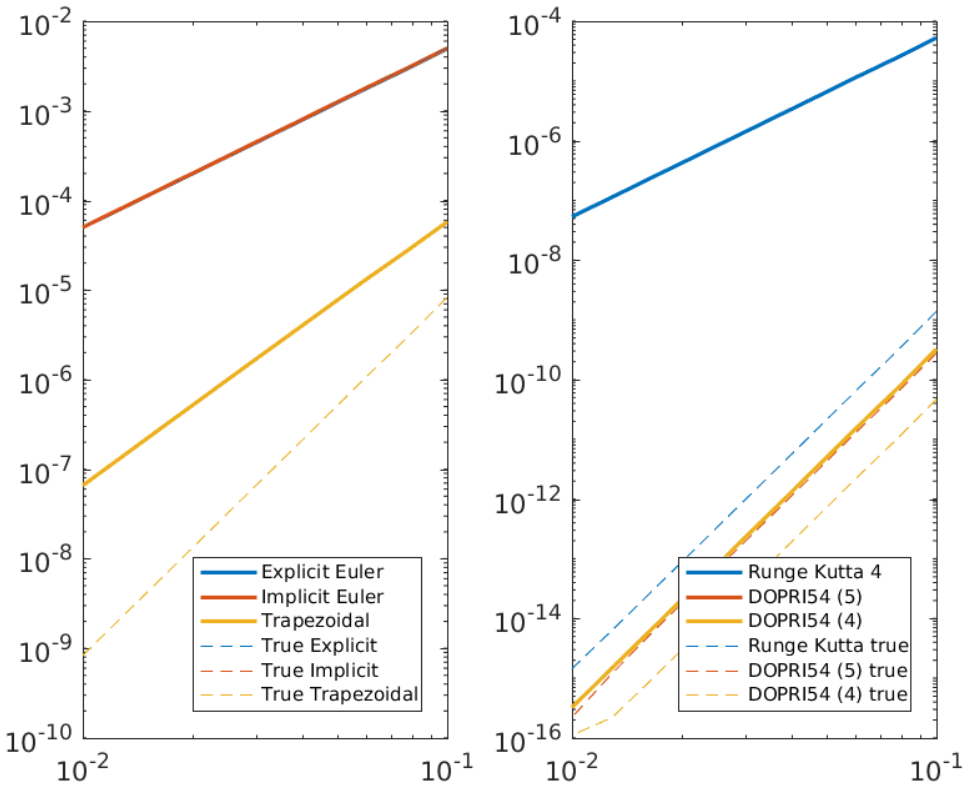

Figure 7: Explicit, Implicit Euler and Trapezoidal method local error estimates in harmonic oscillator

2 The Van der Pol System

In this section we are going to test the implemented methods using the Van der Pol oscillator. This particular initial value problem is very useful to evaluate how numerical solvers respond to stiffness.

Intuituively one can think of stiff problems as those whose solution varies rapidly in a short time span. This characteristic has an strong impact in some numerical methods that might need to take very small steps to get an accurate solution. Since their approximation is based in previous points, explicit methods are specially affected by this phenomena. For these methods, spontaneous variations have an strong effect on the truncation errors and make the numerical solution diverge from the real one. The concept of stiffness is strongly related to the stability of numerical algorithms which is discussed in sections 3 and 5.

Figure 8 shows the solution obtained with the 5 different methods when $\mu = 3$. The graph on the left is the error estimation. In this case, all methods are able to converge to the true solution without having to take very small steps.

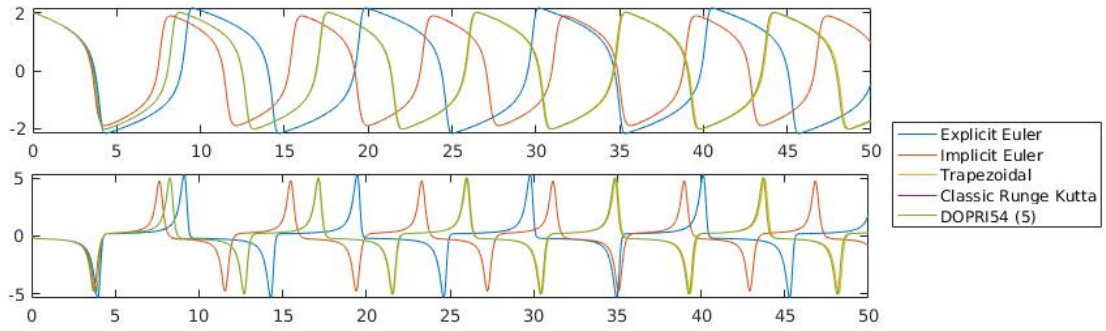

Figure 8: Van der Pol oscillator solution from $t = 0$ to $t = 50$ for $\mu = 3$ and $h = 0.05$

On the other hand, figure 9 shows the solution for $\mu = 100$. For this value of the parameter the problem becomes very stiff in some regions. The bottom graph represents the degree of variation and the peaks give a measure of how fast the solution moves. We see that while the explicit algorithms are unable to handle the problem and their error blows up, implicit methods such as backward Euler can easily follow the solution track because the rely on future values and thus, they are aware in advance of those rapid variations.

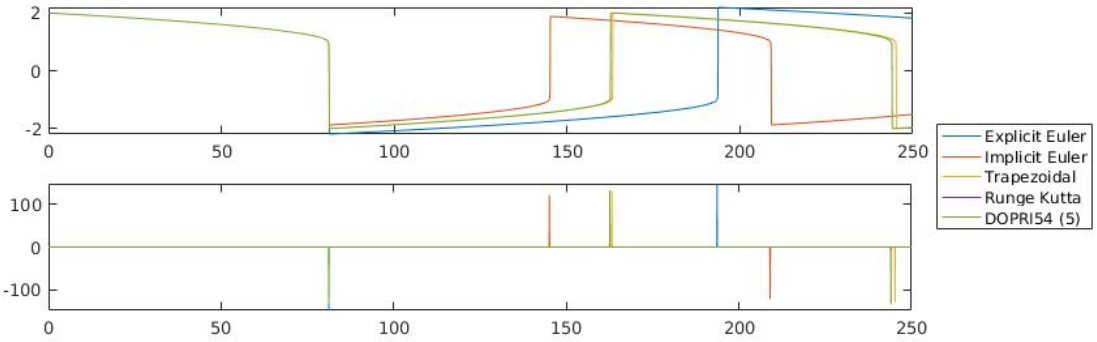

Figure 9: Van der Pol oscillator solution from $t = 0$ to $t = 250$ for $\mu = 100$ and $h = 0.0025$

We have measured the local error at $t = t_0 + h$ for different step sizes and $\mu = 100$. Figure 10 shows that the implicit methods, that is, Implicit Euler and

trapezoidal are accurate even when taking large steps. On the other hand, the error of the high order explicit methods becomes extremely large for step sizes of 10^-1. The fact that the trapezoidal method is an hybrid of the forward and the backward finit difference approximations explains why implicit Euler perfoms better for large steps.

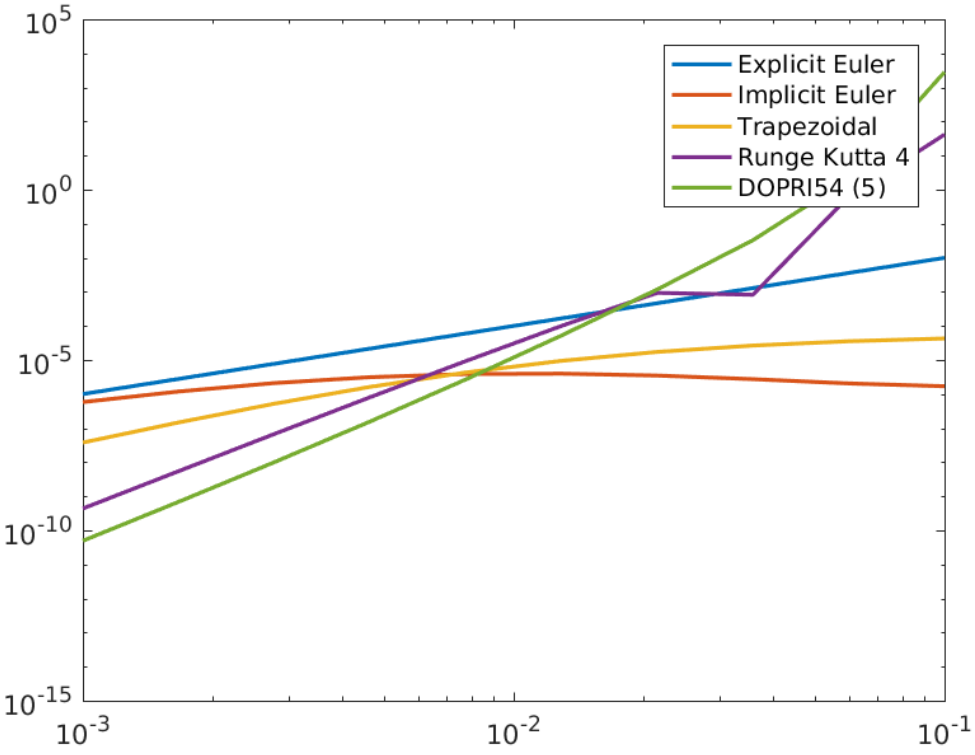

Figure 10: Van der Pol oscillator local error estimates for $\mu = 100$

3 Question 3

3.1 Order conditions, coefficients for the error estimator and the Butcher tableau

Using the excerpt from the book provided in the lecture 10 folder we will write up the order conditions for an embedded Runge-Kutta method with 3 stages. The solution will have order 3 and the embedded method used for error estimation will have order 2.

Firstly the Butcher tableau for our ERK will have the following schema (henceforth the upper triangular shape where the a_{ij} coefficients are 0 and $c_1 = 0$):

0	0	0	0
c_2	a_{21}	0	0
c_3	a_{31}	a_{32}	0
x	b_1	b_2	b_3
$\widehat{x}$	$\widehat{b}_1$	$\widehat{b}_2$	$\widehat{b}_3$
e	d_1	d_2	d_3

Table 1: Butcher tableau for ERK with 3 stages and embedded method

Order conditions (one for first order, one for second order and two for third order) derived from our Butcher tableau:

$$\mathcal{O}(h^1): \quad b^T e = 1 \quad b_1 + b_2 + b_3 = 1 \quad \tau_1 \to \bullet \tag{9a}$$

$$\mathcal{O}(h^2): \quad b^T C e = \frac{1}{2} \quad \underbrace{b_1 c_1}_{0} + b_2 c_2 + b_3 c_3 = \frac{1}{2} \quad \tau_2 \to \vdots \tag{9b}$$

$$\mathcal{O}(h^3): \quad b^T C^2 e = \frac{1}{3} \quad \underbrace{b_1 c_1^2}_{0} + b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \quad \tau_3 \to \therefore \tag{9c}$$

$$b^T A C e = \frac{1}{6} \quad \underbrace{b_2 a_{21} c_1}_{0} + \underbrace{b_3 a_{31} c_1}_{0} + b_3 a_{32} c_2 = \frac{1}{6} \quad \tau_4 \to \vdots \tag{9d}$$

values of c_2 and c_3 will be set to $\frac{1}{4}$ and 1 respecively. This leaves us with 6 unknown variables (3 as and 3 bs) and only 4 equations so we will add the so called consistency conditions in order for the system to be solvable.

$$c_2 = a_{21} \tag{9e}$$

$$c_3 = a_{31} + a_{32} \tag{9f}$$

Using Matlab to solve the system we get the following results:

$$b_1 = -\tfrac{1}{6},\ b_2 = \tfrac{8}{9},\ b_3 = \tfrac{5}{18},\ a_{21} = \tfrac{1}{4},\ a_{31} = -\tfrac{7}{5},\ a_{32} = \tfrac{12}{5}.$$

Next we will solve the system defined for second order embedded method with one first order and one second order condition where c_2 and c_3 are known thus giving 2 equations with 3 unknowns. In order to find a solution, $\widehat{b}_2$ is set to

be $\frac{1}{2}$[1].

$$\widehat{b}_1 + \widehat{b}_2 + \widehat{b}_3 = 1 \tag{10a}$$

$$\widehat{b}_2 c_2 + \widehat{b}_3 c_3 = \frac{1}{2} \tag{10b}$$

The above system yields $\widehat{b}_1 = \frac{1}{8}$ and $\widehat{b}_3 = \frac{3}{8}$. Going back to the Butcher tableau we know that last row $e = (d_1, d_2, d_3)$ is just the difference of the previous two rows by definition.

$c_1 = 0$	0	0	0
$c_2 = \frac{1}{4}$	$1/4$	0	0
$c_3 = 1$	$-7/5$	$12/5$	0
x	$-1/6$	$8/9$	$5/18$
$\widehat{x}$	$1/8$	$1/2$	$3/8$
e	$-7/24$	$7/18$	$-7/72$

Table 2: Butcher tableau with error estimators for our method

[1]According to the book excerpt given in Lecture 10 folder. Otherwise any real value < 1 could have been selected.

3.2 Testing on the test equation

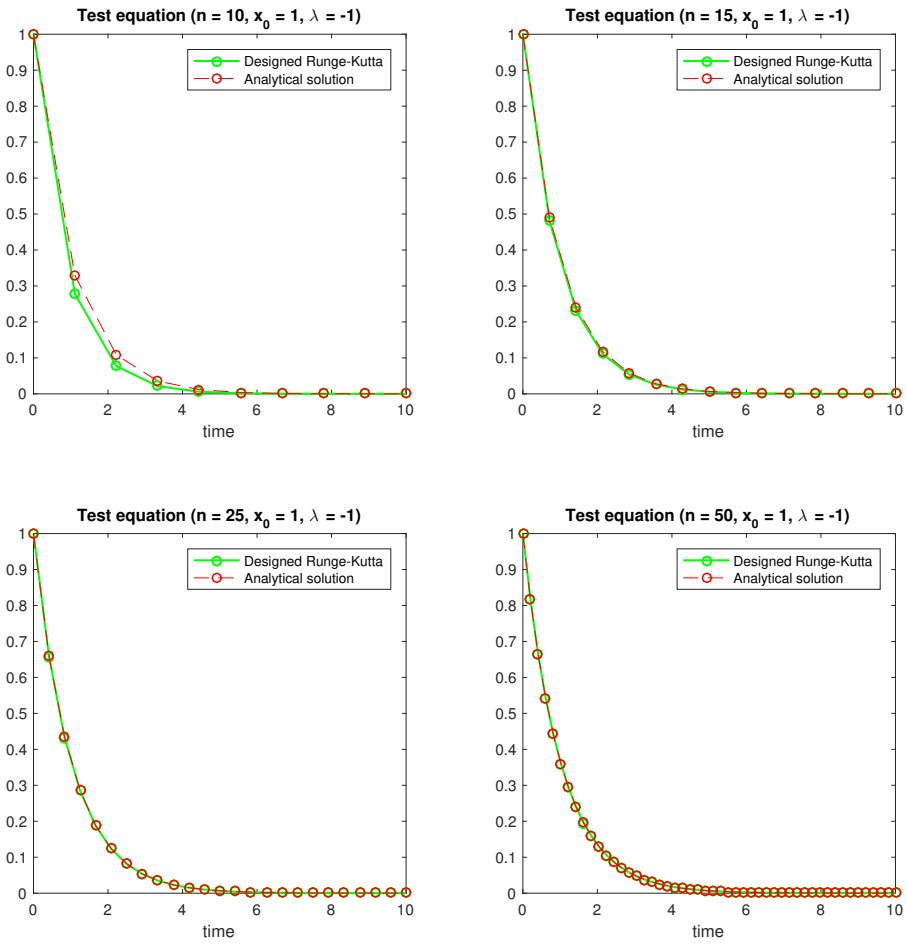

Figure 11: Comparison with the test equation for different step sizes

3.3 Verifying the order

3.4 R(λh) and stability plot

The solution to the test equation obtained by a Runge-Kutta method is defined as $x(t_n + h) = R(\lambda h)x(t_n)$ and $R(z) = 1 + zb^T(I - zA)^{-1}e$. From the Butcher tableau with error estimators for our method vector b and the A matrix are plugged in to

$R(z)$ resulting in

$$R_m(z) = 1 + z + \frac{1}{2}z^2 + \frac{3}{18}z^3$$

where $z = \lambda h$ for the third order method. The second order embedded method yields

$$R_e(z) = z + \frac{1}{2}z^2 + \frac{9}{40}z^3$$

where $z = \lambda h$. Note that $R(z)$ can be calculated with Matlab's Symbolic Toolbox
`syms z;`
`R = 1 + z*b'*inv(eye(length(b)) - z*A)*ones(length(b),1);`
then `collect(R, z)` is used display powers of z and the respective coefficients.

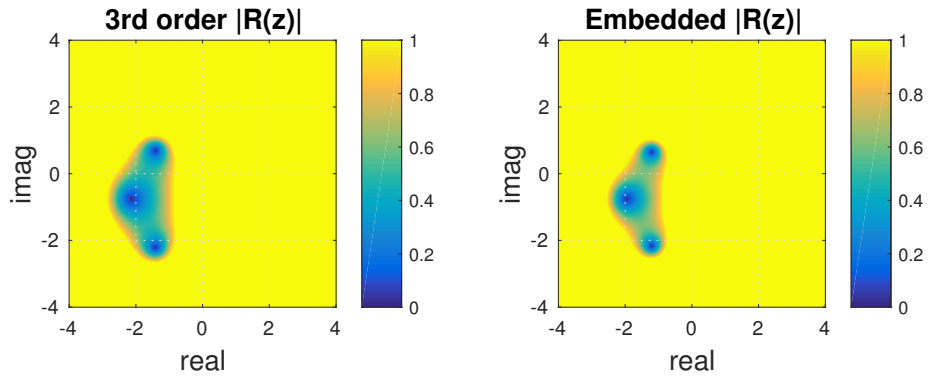

Figure 12: Stability plots of the third order ERK with second order embedded method

3.5 Testinging on the Van der Pol problem and comparison with `ode15s`

Matlab's `ode15s` was used with the default ODE-options and user defined Jacobian, the error for our method with with step size of 10^{-3} is roughly around 10^{-8} and for step size 10^{-2} it is around 10^{-5}. Even though our choice of $c_2 = 1/4$ might look strange, the method performs reasonably well.

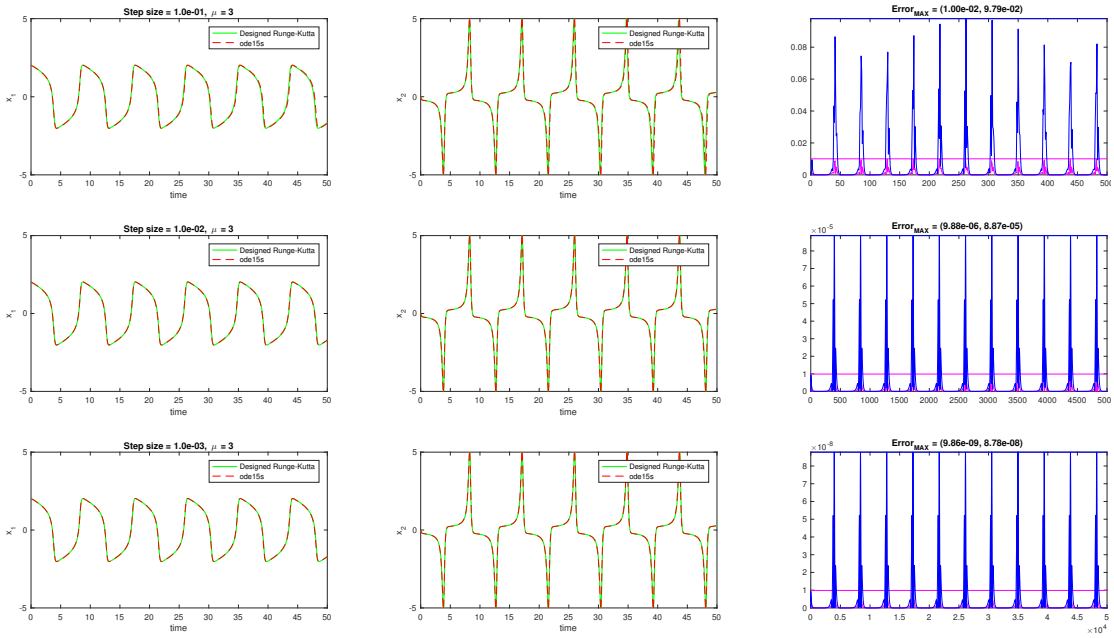

Figure 13: Comparison with `ode15s` on Van der Pol problem ($\mu = 3$). Each row depicts different step size (0.1, 0.01 and 0.001) and the maximal error from ERK is shown in the plot title as well as on a reference line (x_1 - magenta, x_2 - blue).

4 Step size controller

Ideally we would like to be able to obtain an accurate solution without having to take too many unnecesary steps. With this intention in mind we are going to modify the code, so that instead of having a fixed step size, we will give our method the ability to increase or decrease it depending on the error estimation.

Since we already know how to estimate the local error (step doubling or embbeded methods) we can check whether the error made for a specific step size is below a given tolerance. In case the error is larger than this tolerance, we reject the step and we update its size.

There are many ways one can control and update the step size. In our case we will make use of asymptotic and PI controllers. The former makes the update taking into account just the current value of the error, while the latter also relies on the previous estimation.

We have tested the five different methods for different values of the absolute and relative tolerances. The results are gathered in table **??**

Absolute tol $= 10^{-3}$ Relative tol $= 10^{-3}$			
Method	Evaluations	Accepted	Rejected
Explicit Euler	3175	1507	163
Implicit Euler	21528	1490	186
Trapezoidal	18959	504	199
Runge-Kutta 4	10949	650	381
DROPI54	1749	178	0

Table 3: Number of function evaluations number of steps accepted and number of steps rejected using an asymptotic controller

Absolute tol $= 10^{-5}$ Relative tol $= 10^{-5}$			
Method	Evaluations	Accepted	Rejected
Explicit Euler	29847	14922	5
Implicit Euler	139588	14860	5
Trapezoidal	47931	2190	133
Runge-Kutta 4	67155	6056	55
DROPI54	3487	380	0

Table 4: Number of function evaluations number of steps accepted and number of steps rejected using an asymptotic controller

Absolute tol $= 10^{-5}$ Relative tol $= 10^{-5}$			
Method	Evaluations	Accepted	Rejected
Explicit Euler	29856	14922	14
Implicit Euler	139674	14860	15
Trapezoidal	54563	2202	443
Runge-Kutta 4	69431	6062	276
DROPI54	5734	383	0

Table 5: Number of function evaluations number of steps accepted and number of steps rejected using a PI controller

Absolute tol $= 10^{-8}$ Relative tol $= 10^{-8}$			
Method	Evaluations	Accepted	Rejected
Explicit Euler	943470	471728	17
Implicit Euler	4621862	730992	65
Trapezoidal	350341	24438	367
Runge-Kutta 4	2118392	192553	32
DROPI54	13772	1389	0

Table 6: Number of function evaluations number of steps accepted and number of steps rejected using a PI controller

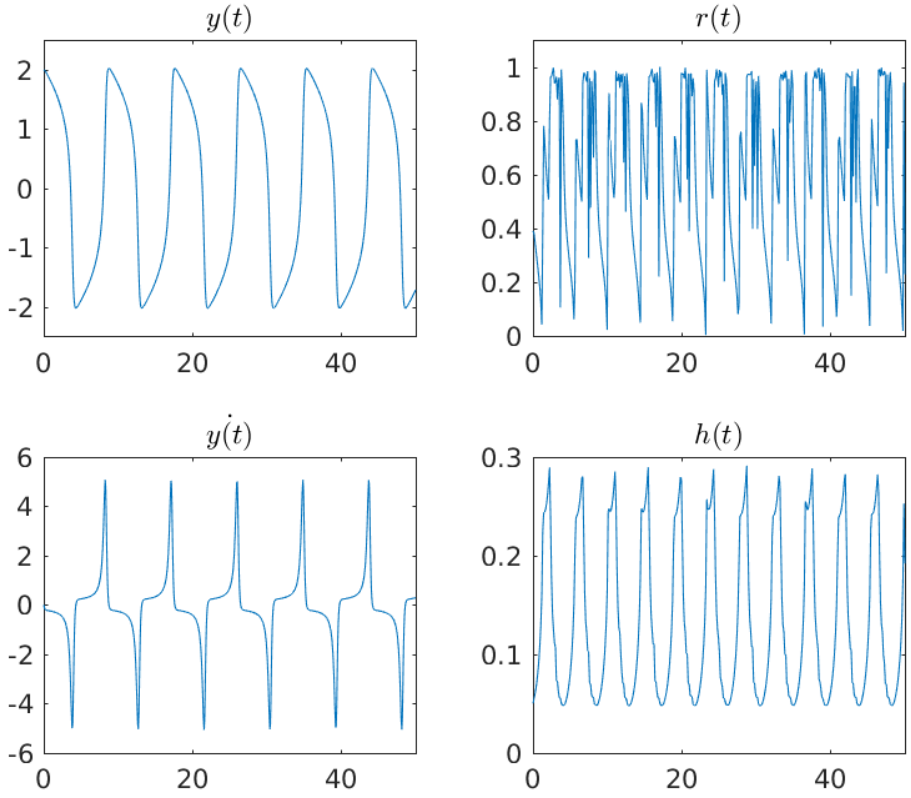

Figure 14: Solution of the Van der Pol oscillator for DOPRI 54 with adaptive step size