Project 1

Numerical Methods for Differential Equations

Isabelle Frodé och Sara Enander
Date: 19 november 2019

1 Explicit Runge-Kutta Methods

The first part of the project aim to construct an ODE-solver based on Runge-Kutta methods. The ODE-solver adjust the stepsize to keep the estimated error at a specific chosen tolerance level, estimating the local error after every new step taken.[2] The method is convenient for approximations of nonstiff problems. As problems get stiffer the method require an increasingly amount of time to compute the calculations and you reach a point when the method no longer works.[1]

The first step to create the ODE-solver was to write a function that takes a single step with the "classic Runge-Kutta method", the RK4 method. The method computes four stage derivatives Y'_1, Y'_2, Y'_3, Y'_4 by sampling f(t,y) at four points, as presented in Equation 1, and then creating an average derivative, shown in Equation 2.[1]

$$\begin{aligned} Y_1' &= (t_n, y_n) \\ Y_2' &= (t_n + h/2, y_n + hY_1'/2) \\ Y_3' &= (t_n + h/2, y_n + hY_2'/2) \\ Y_4' &= (t_n + h, y_n + hY_3') \end{aligned} \tag{1}$$

$$y_{n+1} = \frac{h}{6}(Y_1' + 2Y_2' + 2Y_3' + Y_4') \tag{2}$$

Equation 3, known as the linear test equation [3], was then used to test the function.

$$y' = \lambda y \tag{3}$$

Figure 1 shows the global error for the method applied on the linear test equation, with $\lambda = -1$. By calculating the k-value of the curve (the slope of the curve) in the loglog plot it was confirmed that the global error is indeed $O(h^4)$ on the interval $h \in [10^{-2}, 1]$. Outside this interval, stability problems presumably occur. This is shown in Figure 1 for h larger than approximately 1. The figure however, shows that the implementation is correct, since the RK4 method always have a total accumulated error of $O(h^4)$.[2]

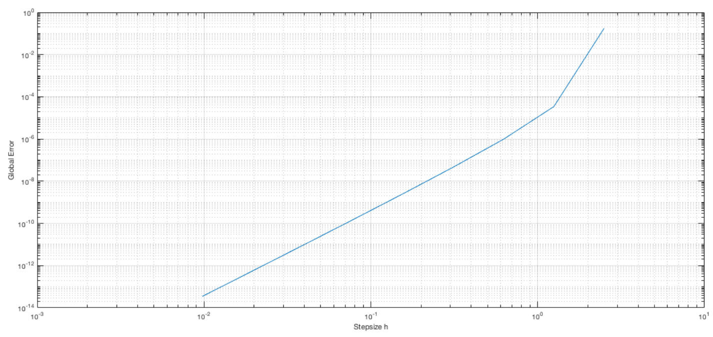


Figure 1: Loglog plot, global error as a function of stepsize

For the next step in the process, an embedded pair of RK methods was created, with an extra evaluation for the solver to be able to estimate a local error. This new RK34 method takes a step with the RK4 method and estimate the local error with the RK3 method. The local error and the result is then stored in the function. The formula for the fifth evaluation is presented in Equation 4, and the estimation of the local error is presented in Equation 5.[1]

$$Z_3' = f(t_n + h, y_n - hY_1' + 2hY_2') \tag{4}$$

$$l_{n+1} = \frac{h}{6}(2Y_2' + Z_3' - 2Y_3' - Y_4') \tag{5}$$

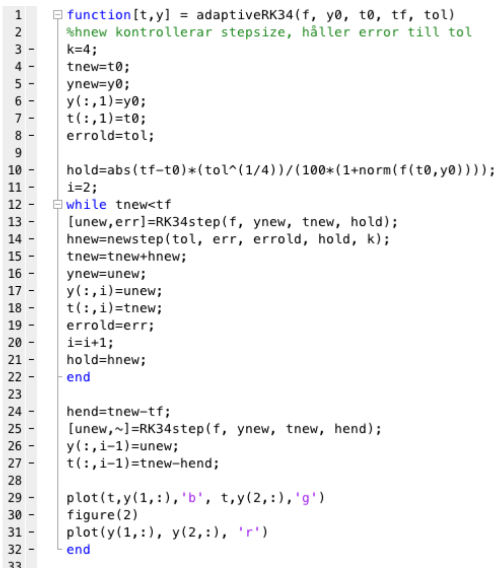
To make the method adaptive and keep the estimated error constant at a specific given tolerance level, a function was created to adjust the stepsize. The new stepsize was computed using a proportional-integral controller which uses the current and the previous error estimate to calculate an error value based on proportional and integral terms. The formula of the PI-controller is presented in equation 6, where r$_n$ and r$_{n-1}$ are the errors estimated, and tol the maximum error tolerance.[1]

$$h_n = \left(\frac{tol}{r_n}\right)^{2/(3k)} \left(\frac{tol}{r_{n-1}}\right)^{-1/(3k)} \cdot h_{n-1} \tag{6}$$

The ODE-solver was constructed by combining the functions described above. The idea is that the solver (starting at a given value, with a given tolerance level), takes a step with RK34 function, returning the local error and the approximated value and then evaluates the next stepsize. This loop run until the next t is larger than what is permitted within the interval. The last stepsize is then created to fit the interval perfectly. Since the algorithm is dependent on the previous stepsize, the stepsize was recommended in the Project manual to start at a value calculated by Equation 7.[1]

$$h_0 = \frac{|t_f - t_0| \cdot tol^{1/4}}{100 \cdot (1 + \|f(y_0)\|)} \tag{7}$$

The created ODE-solver solver is presented in Figure 2. Associated functions, called for in the ODE-solver is presented in Appendix 1.



```
function[t,y] = adaptiveRK34(f, y0, t0, tf, tol)
%hnew kontrollerar stepsize, håller error till tol
k=4;
tnew=t0;
ynew=y0;
y(:,1)=y0;
t(:,1)=t0;
errold=tol;

hold=abs(tf-t0)*(tol^(1/4))/(100*(1+norm(f(t0,y0))));
i=2;
while tnew<tf
[unew,err]=RK34step(f, ynew, tnew, hold);
hnew=newstep(tol, err, errold, hold, k);
tnew=tnew+hnew;
ynew=unew;
y(:,i)=unew;
t(:,i)=tnew;
errold=err;
i=i+1;
hold=hnew;
end

hend=tnew-tf;
[unew,~]=RK34step(f, ynew, tnew, hend);
y(:,i-1)=unew;
t(:,i-1)=tnew-hend;

plot(t,y(1,:),'b', t,y(2,:),'g')
figure(2)
plot(y(1,:), y(2,:), 'r')
end
```

Figure 2: Implementation of the Adaptive ODE-solver

2 Nonstiff Lotka-Volterra Problem

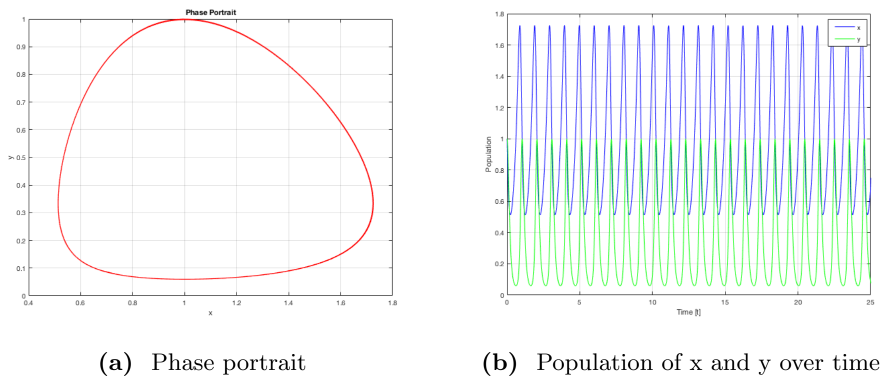
In this section, a nonstiff problem was studied using equation 8, the Lotka-Volterra equation, which is a model for biological population dynamics between prey and predator.[1]

$$\begin{aligned} \dot{x} &= ax - bxy \\ \dot{y} &= cxy - dy \end{aligned} \tag{8}$$

The parameters a, b, c, d were chosen as (9, 9, 15, 15). Using the ODE-solver created in the previous task, an approximation of the solution could be computed. To prove the periodicity of the solution, a phase portrait was plotted and presented in Figure 3 (a), starting off with the initial values x(0)= 1, y(0)=1. The figure is showing a plot of a closed orbit, hence the solution is periodic.

To determine the length of a period another graph was studied. Figure

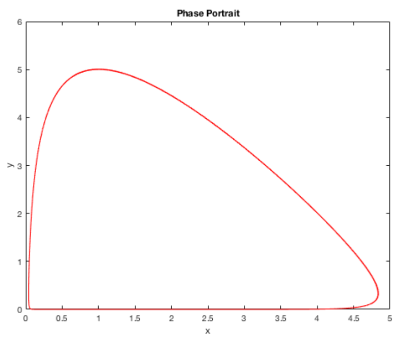
3 (b) shows the solutions x and y as functions of time. The length of one period was then approximated through counting the total number of periods shown in the graph. The time (given on the x-axis) divided by the total number of periods, was then calculated to be approximately 1.02 time units.



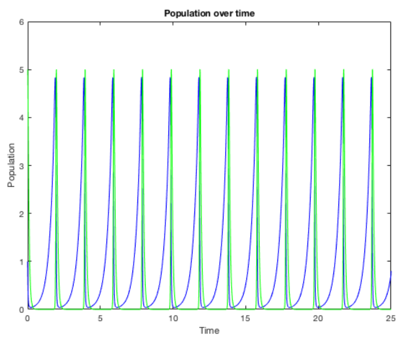
(a) Phase portrait **(b)** Population of x and y over time

Figure 3: Solutions to Lotka-Volterra equation, initial conditions y(0)=1, x(0)=1

When the initial conditions was changed, periodicity was not affected if the initial values was within the interval of stability. If however, for example, one of the initial values was zero, then the solution would fade out directly and the solutions would not be periodic. But when the initial conditions was set to be x(0) = 1, y(0) = 5, and x(0) = 5, y(0) = 1, the solutions were still periodic. Figure 4 (a) shows the phase portrait when the initial values are x(0) = 1, y(0) = 5 (the other one is similar). The plot (in both cases) is a closed orbit, hence the solution is periodic. The length of one period however, does not remain the same when the initial conditions are changed. Figure 4 (b) shows the population y and x over time. One period in this plot is calculated to be approximately 1.92 seconds, which is longer than what we had for the initial values x(0)=1, y(0)=1.



(a) Phase portrait



(b) Population of x and y over time

Figure 4: Solutions to Lotka-Volterra equation, initial conditions y(0)=1, x(0)=5

By first dividing the Lotka-Volterra equation and then separate them we get a new function H, shown in equation 9, whereas H is a function of both x and y[1].

$$H(x, y) = cx + by - d \cdot log(x) - a \cdot log(y) \tag{9}$$

To investigate whether the numerically computed H is invariant along solutions, it was integrated over a long time to check if H remains constant. Figure 5 shows equation 10[1]

$$|\frac{H(x, y)}{H(x(0), y(0))} - 1| \tag{10}$$

as a function of time. It is expected that the function will stay near zero if H stays near its initial value.

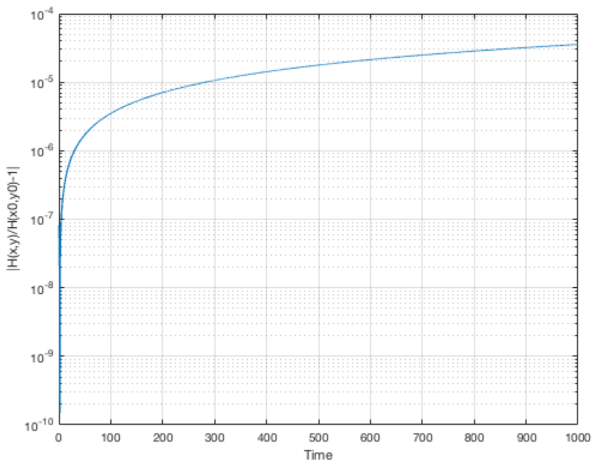


Figure 5: $|\frac{H(x,y)}{H(x0,y0)} - 1|$ plotted as a function of time

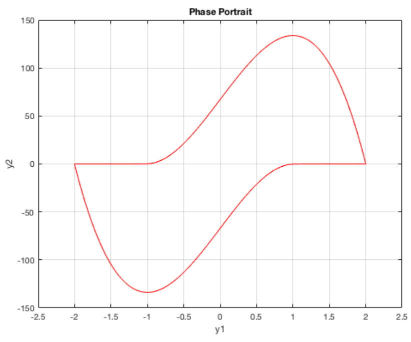
Figure 5 shows that H stays near its initial value when time increases, since the difference is less than 10^{-4} in the plot. Even though the ratio increases as time increases, it does not go beyond 10^{-4}. The linlog-plot shows that H(x,y) stays near its initial values and does not drift away.

3 Stiff and Nonstiff Problems

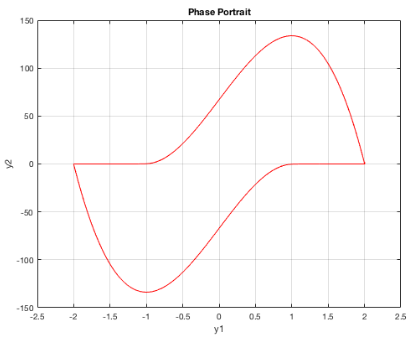
The first task of this section was to solve the van der Pol equation[1]:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \mu \cdot (1 - y_1^2) \cdot y_2 - y_1 \end{aligned} \tag{11}$$

It was solved for μ=100, on the time interval [0, 2μ] using the ODE-solver created in section 1, presented in Figure 2. The tolerance was set to tol = 10^{-6}. To check the periodicity, phase portraits were plotted for different initial values. The phase portraits for $y_1(0)=0$, $y_2(0)=2$ and $y_1(0)=2$, $y_2(0)=2$ is shown in Figure 6. The plots are showing the limit cycle, in other words, the same closed orbit for different initial values. The conclusion is that when the problem is stable, the solution to the van der Pol equation tend to the same oscillation.



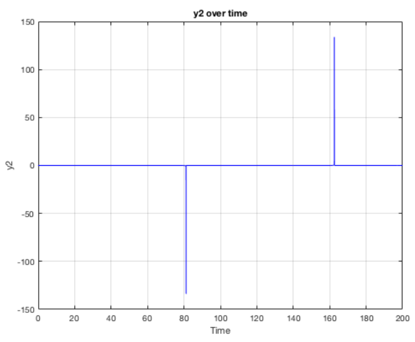
(a) Initial values $y_1(0)=0$ $y_2(0)=2$



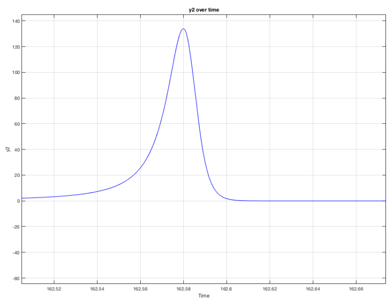
(b) Initial values $y_1(0)=2$ $y_2(0)=2$

Figure 6: Phase portraits of the solution of the van der Pol equation

The approximated solution was plotted in Figure 7. The solution, shown in 7 (a), may at first sight appear as a solution to a stiff problem. However, if one look closer as shown in Figure 7 (b), the plot is clearly showing a solution to a nonstiff problem.



(a) General solution



(b) Zoomed in on the peak.

Figure 7: Solution to the van der Pol equation for μ=100, y2 as a function of time

To further explore the stiffness of the problem and how it depends on μ, the solutions on the interval $[0,\ 0.7\mu]$ was plotted using the ODE-solver created in the first section. The initial values was set to $y(0) = (2\ 0)^T$ for all computations and the problem was solved for μ={10, 15, 22, 33, 47, 68, 100, 150, 220, 330, 470, 680, 1000}. The tolerance was set to tol $= 10^{-7}$. The total number of steps, N, the ODE-solver needs to complete the integration for each μ was plotted as a function of μ. The result is presented in Figure 8. The script for this computation is found in Appendix 2. Figure 8 is showing a loglog-plot with the slope equal or almost equal to 2. That tells us that the total number of steps N to complete an integration $\propto \mu^q$, whereas q $\simeq$ 2. N $\propto$ the stiffness, hence the stiffness $\propto \mu^2$.

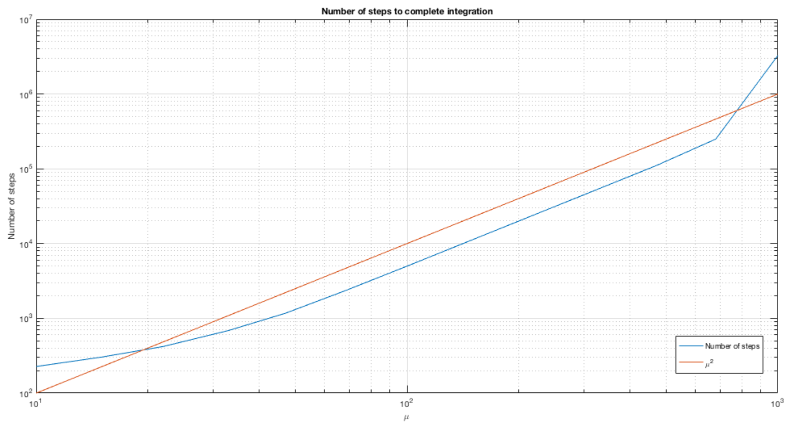


Figure 8: Number of steps as a function of μ, with an orange reference line with slope 2

The solver created in previous task was then compared with MATLAB's own solver for stiff problems, ode15s. The experiment above was conducted again, but this time with the ode15s solver. The difference between the solutions is shown in figure 9:

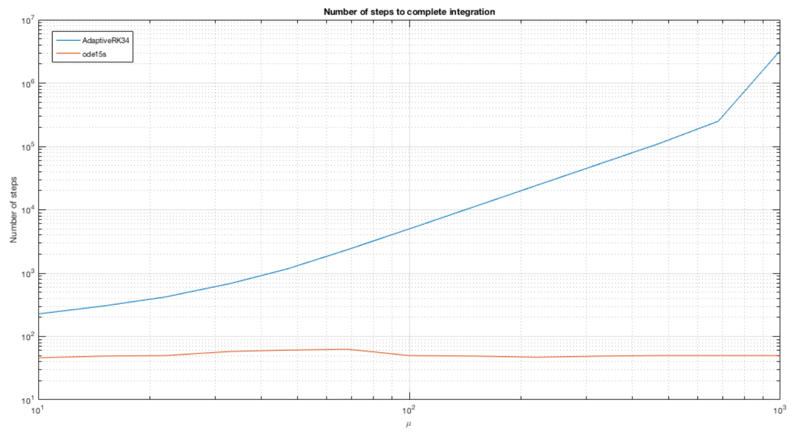


Figure 9: Number of steps as a function of μ, comparison between the adaptiveRK34 and MATLAB's solver ode15s

The plot shows that MATLAB's solver always requires fewer steps than the solver created in section 1, which indicates that MATLAB's solver works better than the adaptiveRK34. The number of steps is also almost constant, even when μ is higher and the adaptiveRK34 gets stability problems. This might be because MATLAB's solver is constructed for stiff problems, and the adaptiveRK34 for nonstiff problems.[1] Since higher μ means stiffer problem, it is expected that the ode15s works much better when μ is big. Then μ was set to 10,000 and solved with ode15s, solution shown in figure 10. The time it took to solve it was not noticeable longer than for smaller μ. The plot looks the same, but with a factor 100 difference, which was expected since μ was set 100 times bigger.

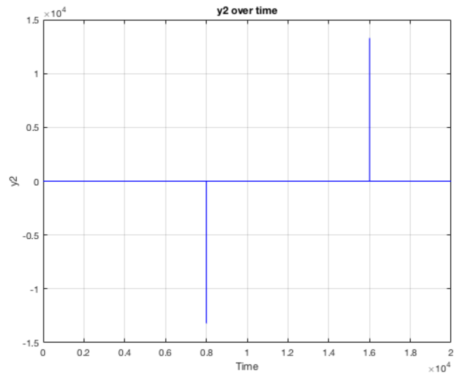


Figure 10: Solution when μ was set to 10 000

4 Project Summary

In this project, we definitely got started with MATLAB while approximating solutions to initial value ODEs. Runge-Kutta methods were used to implement our own ODE-solver and Lotka-Volterra- and van der Pol problems could be solved. Stiffness of the problems were studied and we learned that our solver got worse as the problem got stiffer.

Getting stuck for a very long time on the first task taught us to use the debugger function in MATLAB and above all, the importance of patience and grit when coding. When working with the project, we did two separate scripts and then compared them frequently. The report was then written together.

Last but not least, we would like to acknowledge Julio and Tony who helped us when getting stuck with our code and also a big thank you to our fellow classmates Louise Karsten, Anna Sjerling, Felicia Segui and Ebba Rickard for great brainstorming and inspirational fika pauses.

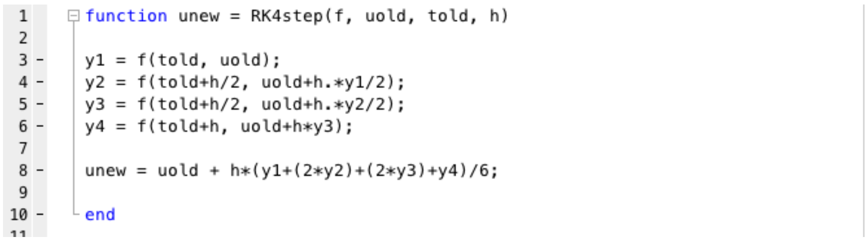
References

[1] T Stillfjord, G Söderlind, Project manual: *Project 1 in FMNN10 and NUMN12*, LTH, 2019

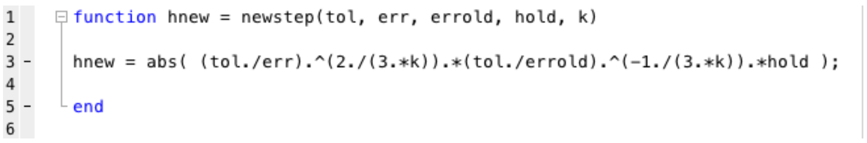
[2] T Stillfjord, *Lecture Notes Chapter 1*, https://canvas.education.lu.se/courses/902/files/127269/download?download_frd=1, LTH, 2019

[3] T Stillfjord, *Lecture Notes Chapter 2*, https://canvas.education.lu.se/courses/902/files/131097/download?download_frd=1, LTH, 2019

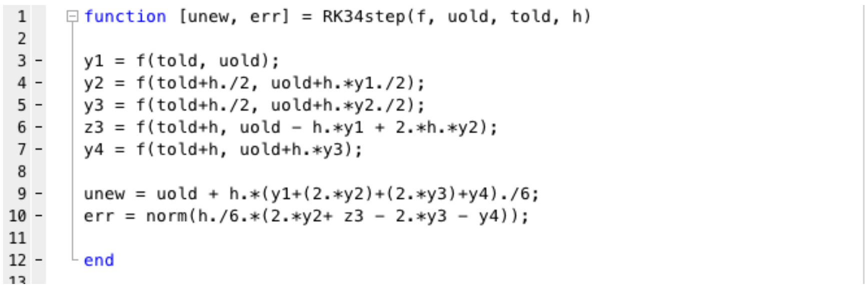
5 Appendix 1



```matlab
function unew = RK4step(f, uold, told, h)

y1 = f(told, uold);
y2 = f(told+h/2, uold+h.*y1/2);
y3 = f(told+h/2, uold+h.*y2/2);
y4 = f(told+h, uold+h*y3);

unew = uold + h*(y1+(2*y2)+(2*y3)+y4)/6;

end
```

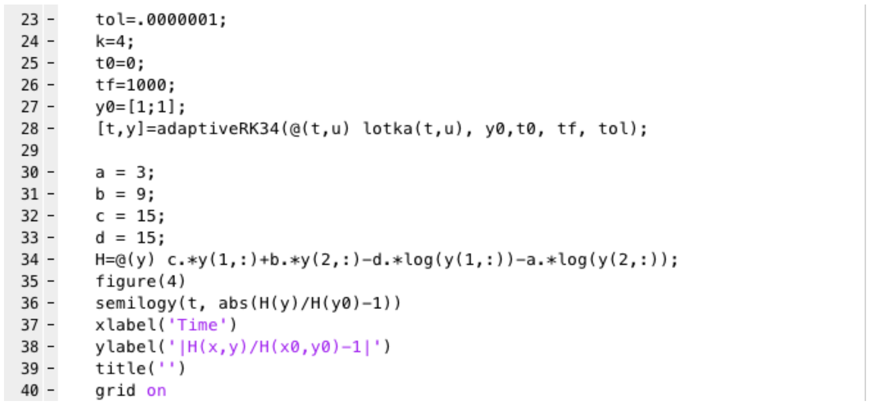


```matlab
function hnew = newstep(tol, err, errold, hold, k)

hnew = abs( (tol./err).^(2./(3.*k)).*(tol./errold).^(-1./(3.*k)).*hold );

end
```



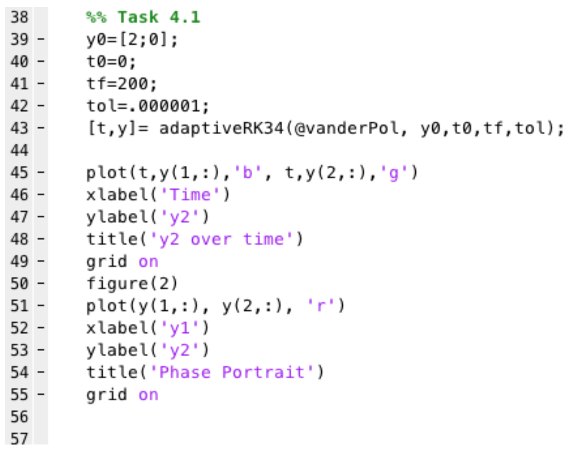
```matlab
function [unew, err] = RK34step(f, uold, told, h)

y1 = f(told, uold);
y2 = f(told+h./2, uold+h.*y1./2);
y3 = f(told+h./2, uold+h.*y2./2);
z3 = f(told+h, uold - h.*y1 + 2.*h.*y2);
y4 = f(told+h, uold+h.*y3);

unew = uold + h.*(y1+(2.*y2)+(2.*y3)+y4)./6;
err = norm(h./6.*(2.*y2+ z3 - 2.*y3 - y4));

end
```

6 Appendix 2

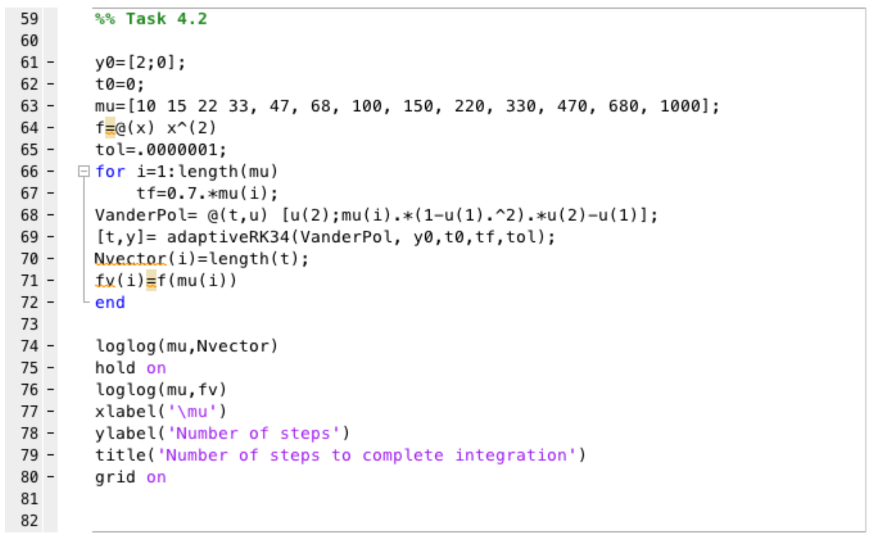


```
tol=.0000001;
k=4;
t0=0;
tf=1000;
y0=[1;1];
[t,y]=adaptiveRK34(@(t,u) lotka(t,u), y0,t0, tf, tol);

a = 3;
b = 9;
c = 15;
d = 15;
H=@(y) c.*y(1,:)+b.*y(2,:)-d.*log(y(1,:))-a.*log(y(2,:));
figure(4)
semilogy(t, abs(H(y)/H(y0)-1))
xlabel('Time')
ylabel('|H(x,y)/H(x0,y0)-1|')
title('')
grid on
```

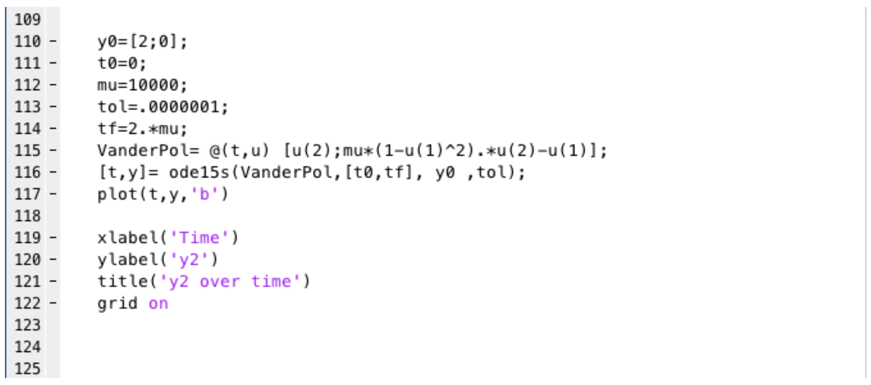
7 Appendix 3



```matlab
%% Task 4.1
y0=[2;0];
t0=0;
tf=200;
tol=.000001;
[t,y]= adaptiveRK34(@vanderPol, y0,t0,tf,tol);

plot(t,y(1,:),'b', t,y(2,:),'g')
xlabel('Time')
ylabel('y2')
title('y2 over time')
grid on
figure(2)
plot(y(1,:), y(2,:), 'r')
xlabel('y1')
ylabel('y2')
title('Phase Portrait')
grid on


```



```matlab
%% Task 4.2

y0=[2;0];
t0=0;
mu=[10 15 22 33, 47, 68, 100, 150, 220, 330, 470, 680, 1000];
f=@(x) x^(2)
tol=.0000001;
for i=1:length(mu)
    tf=0.7.*mu(i);
VanderPol= @(t,u) [u(2);mu(i).*(1-u(1).^2).*u(2)-u(1)];
[t,y]= adaptiveRK34(VanderPol, y0,t0,tf,tol);
Nvector(i)=length(t);
fv(i)=f(mu(i))
end

loglog(mu,Nvector)
hold on
loglog(mu,fv)
xlabel('\mu')
ylabel('Number of steps')
title('Number of steps to complete integration')
grid on


```



```matlab

y0=[2;0];
t0=0;
mu=10000;
tol=.0000001;
tf=2.*mu;
VanderPol= @(t,u) [u(2);mu*(1-u(1)^2).*u(2)-u(1)];
[t,y]= ode15s(VanderPol,[t0,tf], y0 ,tol);
plot(t,y,'b')

xlabel('Time')
ylabel('y2')
title('y2 over time')
grid on



```