Homework2

Ali Fele Paranj
UBC
2021

Table of Contents:

1. Problem1:

In this section I will implement the algorithm that takes one Runge-Kutta steps.

1.1 Theory and Derivation:

Suppose that we want to solve the following differential equation using RK4 method:

$$y^{(n)} = f(x, y, y', y'', ..., y^{(n-1)})$$

To do so, we apply the following change of variable:

$$y_0 = y \qquad y_1 = y' \qquad y_2 = y'' \qquad \dots \qquad y_{n-1} = y^{n-1}$$

So we can change the original differential equation to the following system of first order differential equations:

$$\begin{gathered} y_0' = y_1 = F_0(x, Y(x)) \\ y_1' = y_2 = F_1(x, Y(x)) \\ y_2' = y_3 = F_2(x, Y(x)) \\ \dots \\ y_{n-1}' = y_n = f(x, y, y', y'', , y(n-1)) = F_{n-1}(x, Y(x)) \end{gathered}$$

So all of the equations can be written in the following vector form:

$$Y' = F(x, Y(x))$$

In which:

$$\begin{gathered} Y = (y_0, y_1, y_2, , y_{n-1}) = (y, y', y'', y''', , y^{(n-1)}) \\ F(x, Y(x)) = (F_0, F_1, F_2, , F_{n-1}) \end{gathered}$$

Now to perform the RK4 single step action, we should calculate:

$$Y(x_0 + dx) = Y(x_0) + \frac{h}{6}(f_0 + f_1 + f_2 + f_3)$$

In which:

$$\begin{aligned} f_0 &= F(x_0, Y(x_0)) \\ f_1 &= F(x_0 + dx/2, Y(x_0) + h/2 * f_0) \\ f_2 &= F(x_0 + dx/2, Y(x_0) + h/2 * f_1) \\ f_3 &= F(x_0 + dx, Y(x_0) + h * f_2) \end{aligned}$$

1.2 Code Exploration:

fcn_1.m: This function is the right hand side of the ODE expressed in section 1.3. This is the ODE that we know its exact solution as it is used to calculate the error of RK4step function
rk4step.m: This function implements the single step of Runge-Kutta algorithm. The theory behind this function is discussed in the section above
main1.m: This function is used to set the initial values in order to run the rk4step function. This script is also used to generate the desired outputs.

1.3 Applying to a known ODE:

Now let's apply the rk4step algorithm to an ODE that we know its exact solution. Here I am using the following ODE:

$$\ddot{x} = -x$$
$$x(0) = 0$$
$$\dot{x}(0) = 1$$

We know that the answer of the above ODE is:

$$x(t) = sin(t)$$

So with setting $dt = 0.01$, $t_0 = 0$, lets calculate the value $x(t_0 + dt)$ and $x_{exact}(t_0 + dt)$. The following table summarizes the results.

$x(t_0 + dt)$	$x_{exact}(t_0 + dt)$	$x(t_0 + dt) - x_{exact}(t_0 + dt)$
0.009999833333333	0.009999833334167	0.000000000000833

As you can see, the error is lower than $O(\Delta t^{-5})$ (in this case it is $8 * 10^{-13}$ which is indeed less than $(0.01)^{-5}$

2. Problem2:

In this section, I am going to apply the function derived above to solve the differential equation at multiple steps. To do that I am going to use the variable "tspan" that contains the times steps that I want to find the solution of the differential equation at.

2.1 Code Exploration:

fcn_harmonic.m: This function is the right hand side of the simple harmonic ODE.
fcn_VanDerPol.m: This function is the right hand side of the Van der Pol oscillator.

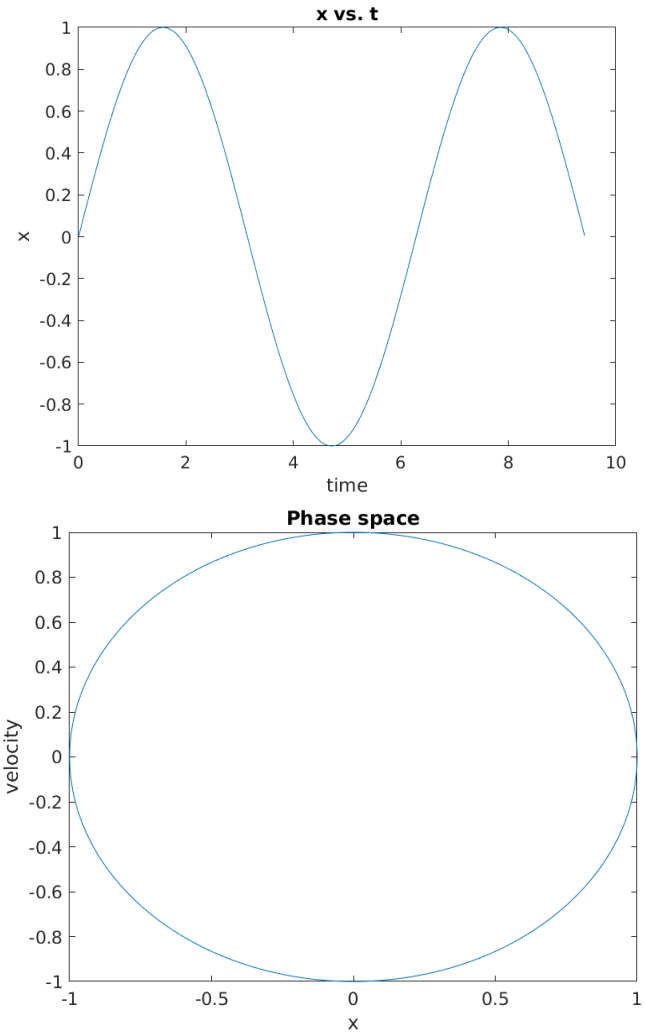
main2.m: This function is used to set the initial values in order to run the rk4 function. This script is also used to generate the desired outputs.

2.2 Simple Harmonic Motion:

In this section I am going to solve the following differential equation (that is for simple harmonic oscillation)

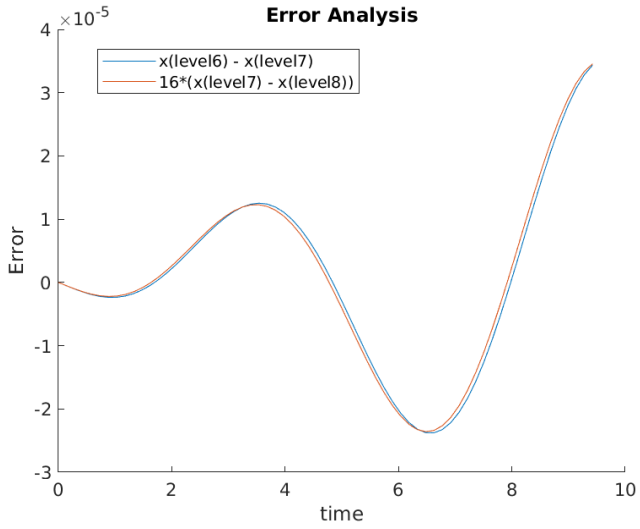
$$\ddot{x} = -x$$
$$x(0) = 0$$
$$\dot{x}(0) = 1$$

Here is some basic outputs of my RK4 solver:



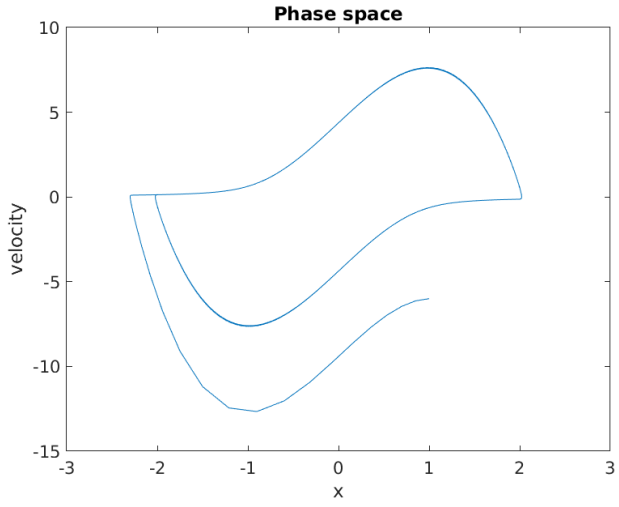
2.2.1 Error Analysis

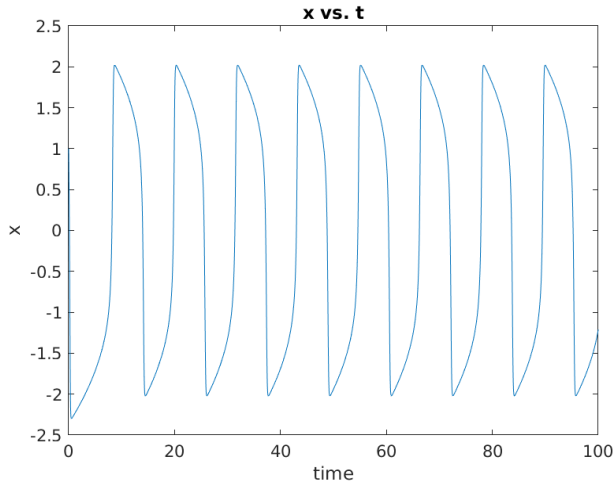
Here in this section I use a three level convergence test to evaluate the error of the RK4 solver. If the RK4 has $O(\Delta t^5)$ error, then if we calculate the values of x at three 6,7,8 level, then the error of $x_{level6} - x_{level7}$ should be 2^4 times bigger than $x_{level7} - x_{level8}$. In the following graph you can easily see this fact.



2.3 Van der Pol:

Now we can use the implemented RK4 solver to solve the Van der Pol ODE. In the following two plots you can see the solved ODE with a=5:





3. Problem3:

In this section I am going to implement the RK4 ODE solver with adaptive step size.

3.1 Theory and Derivation:

As indicated in the home work file, we can write:

$$y_{\Delta t}(t_0 + \Delta t) = y_{exact}(t_0 + \Delta t) + k(t_0)\Delta t^5$$
$$y_{\Delta t/2}(t_0 + \Delta t/2) = y_{exact}(t_0 + \Delta t/2) + k(t_0)(\Delta t/2)^5$$

So we can write:

$$y_{\Delta t/2}(t_0 + \Delta t) \approx y_{exact}(t_0 + \Delta t) + 2 * k(t_0)(\Delta t/2)^5$$

So we can calculate the truncation error as following:

$$\Delta y = |y_{\Delta t/2} - y_{\Delta t}| = k(t_0)\Delta t^5 * 15/16$$
$$E_{trunc}^{\Delta t} = 16/15\Delta y$$

Now suppose that we want to change the step size in a way that the new truncation error would be less than the predetermined error value (E_tol):

$$\Delta t^* = c\Delta t$$
$$E_{trunc}^{\Delta t^*} = c^5 E_{trunc}^{\Delta t} \leq E_{tol}$$

So we can write:

$$C \leq \sqrt[5]{E_{tol}/E_{trunc}^{\Delta t}}$$

As a conservative choice, the following choice of C seems a good option:

$$C = 0.9\sqrt[5]{E_{tol}/E_{trunc}^{\Delta t}}$$

Strategy to implement calculate the ODE at tspan points:

Since I have to calculate the ODE at the times defined at tspan, my ODE solver can not take any time step it wants. So I followed the following strategy to both do a adaptive Runge-Kutta and calculate the ODE at predefined points:

1. First I calculate the Δt according to the adjacent times in tspan list (i.e. $t_0 = tspan(i)$, $\Delta t = tspan(i+1) - tspan(i)$)
2. Then I will calculate the Δt^* to have lower error that predefined E_{tol}
3. If the calculated time step (i.e. Δt^*) is bigger than Δt, I use Δt as the time step
4. But if $\Delta t^* < \Delta t$, then I choose Δt^* as a time step. Then I take $t_0 \leftarrow t_0 + \Delta t^*$. Then I repeat steps 2,3.

3.2. Code Exploration:

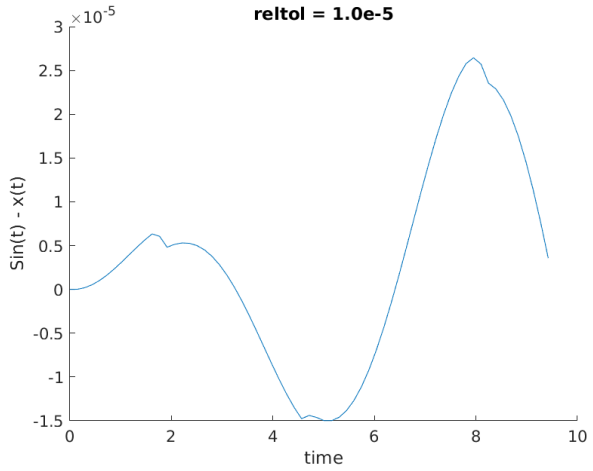
To implement the adaptive Runge-Kutta, I have used two matlab files.

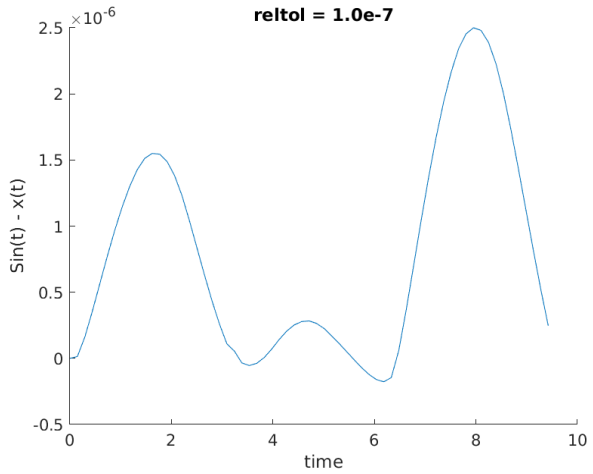
main3.m: This script initialized the values needed to run the adaptive RK algorithm. Also, some parts of this script generate the desired outputs.

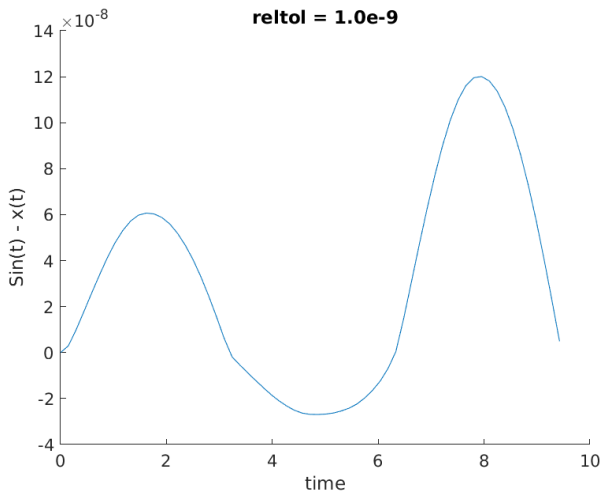
rk4ad.m: This function implements the adaptive RK method. The theory used in this code is described in the section above.

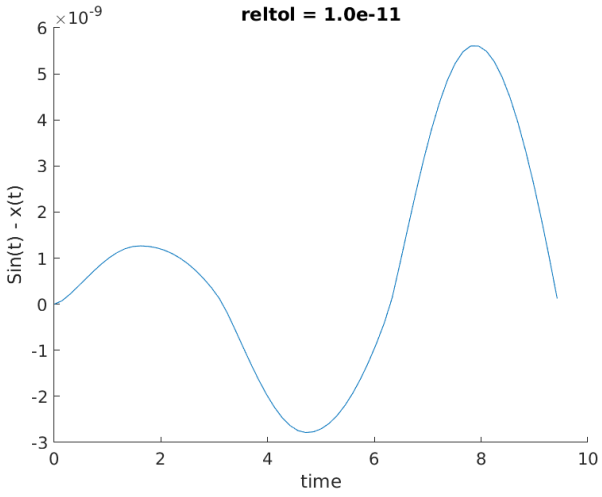
3.3. Harmonic Oscillator:

In this section I run my adaptive RK algorithm to calculate the x(t) with different relative tolerances. We know that the exact solution of the harmonic oscillation with the given initial values at problem 2, will be $x_{exact}(t) = sin(t)$. So to compare the errors, I calculated $sin(t) - x(t)$. In the following plots you can see the output:


reltol = 1.0e-5
×10^-5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
Sin(t) - x(t)
0
2
4
6
8
10
time


reltol = 1.0e-7
×10^-6
2.5
2
1.5
1
0.5
0
-0.5
Sin(t) - x(t)
0
2
4
6
8
10
time

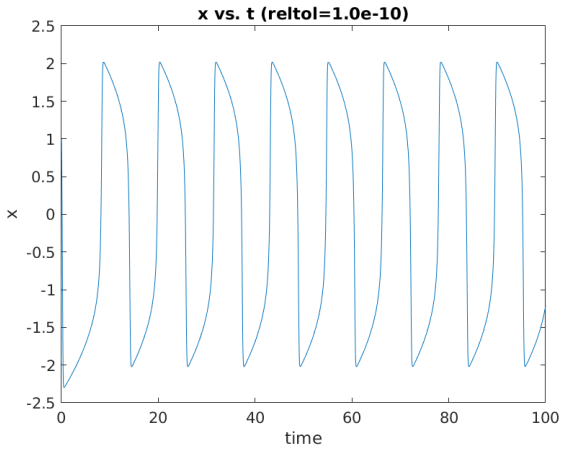

reltol = 1.0e-9
×10^-8
14
12
10
8
6
4
2
0
-2
-4
Sin(t) - x(t)
0
2
4
6
8
10
time

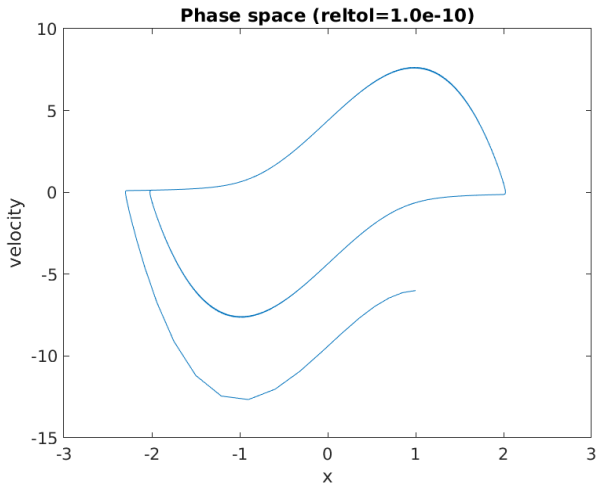


As you can see, the error is always lower than the indicated error.

3.4. Van der Pol Oscillator:

In this section I reproduce the results of the Van der Pol oscillator of problem 2. The relative error has been set to 1e-10. In the following two graphs, you can see the “x vs. t” and “phase space” plots.




Phase space (reltol=1.0e-10)
velocity
x
10
5
0
-5
-10
-15
-3
-2
-1
0
1
2
3