MAB260 Numerical Methods 2 – Coursework 2  
Using 4th-order Runge-Kutta to approximate Numerical Solutions to a system of 1st order Ordinary Differential Equations and the investigation of the “Butterfly Effect” in the Chaos Theory

*Abstract*

Numerical Methods are methods to find numerical approximations to the solution of ordinary differential equations (or known as ODEs). Euler Methods, 2nd and 4th-order Runge-Kutta are numerical methods for solving first-order initial value problems. This report looks at implementation of a generalised 4th-order Runge-Kutta method into a computer algebra program, Maple, computing solution for a nonlinear motion of a simple pendulum (a 2nd order ODE), computing solution for a system of 1st order ODEs, estimating step sizes for ODEs and investigates the “butterﬂy eﬀect”.

*The task/introduction*

The task is to implement a generalised 4th-order Runge-Kutta method into Maple, which could take an initial value problem for a system of N first-order ODEs, to calculate numerical solutions to the two problems.

The first problem is to compute the solution of the equation governing the motion of a pendulum. Then, plotting the angular displacement and velocity against time, plotting the displacement against velocity, comment on the implications of the graph for energy conservation and finding the period of oscillation of the pendulum using the solutions.

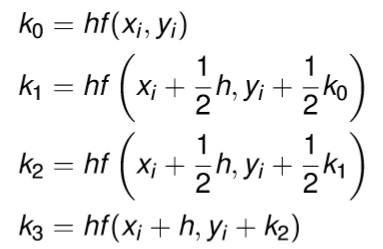
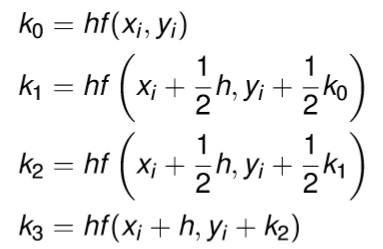
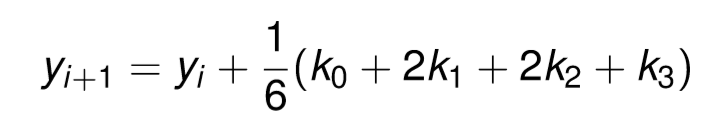
The second problem is to compute the solution to the system of equations which describes the motion of a rigid body when the applied torques depend linearly on the angular velocities about the principal axes of inertia fixed at the centre of mass. Then, plotting the three components of the angular velocity against time and against each other in a three-dimensional phase domain. Finally, changing initial conditions and times to illustrate the “butterfly effect”.

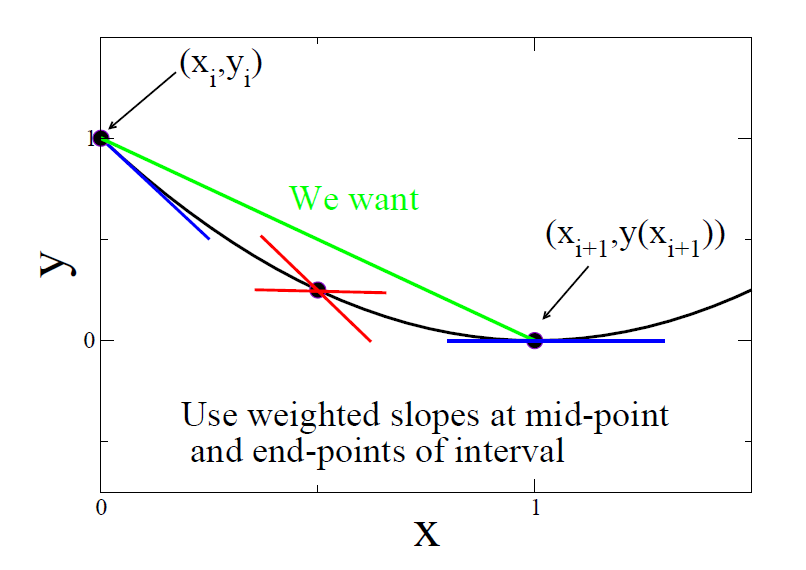
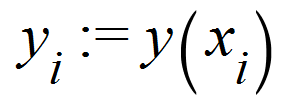
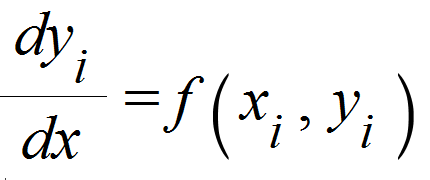
*Content*

* Overview of the 4th-order Runge-Kutta method Page 2
* Method of solutions
  + Generalised 4th-order Runge-Kutta method Page 3
  + Printing the results Page 4
  + Sorting the data points Page 4
  + Plotting graphs Page 4
* Results and Discussion
  + Problem 1: Nonlinear motion of a simple pendulum Page 5
  + Problem 2: Rigid Body Motion Page 9
* Conclusion Page 15
* References Page 15

*Overview of the 4th-order Runge-Kutta method*

4th order Runge-Kutta is a single-step method that are used to find the numerical solution of ordinary differential question (ODEs) initial-value problems. Single-step methods implies the method only uses information from the step that we are presently on, to obtain the result for the next step.   
  
The general form of 4th order Runge-Kutta is as follows:

where  and .

The graph on the left represent the geometric meaning of the 4th-order Runge-Kutta method.

As you can see in the graph, the blue lines represent the average slopes at the endpoints of interval and the red lines represent the two approximation of the slope at the midpoint of interval.

Here, k0, k3 are the blue line from left to right respectively and k1, k2 are the red lines at midpoint.

Using the value k0, k1, k2, k3, we can obtain an approximation for the slopes of the curve x

This methods have a local truncation error O(h5) and a global truncation error O(h4).

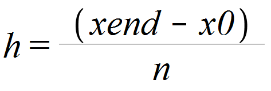
The weakness of this method is the need four function evaluations per step but compensate for this by allowing larger step sizes to be employed.

*Method of solutions*

*Generalised 4th-order Runge-Kutta method*

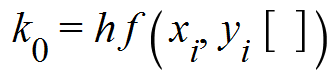
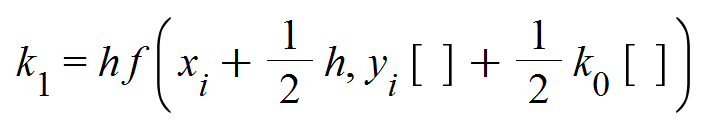
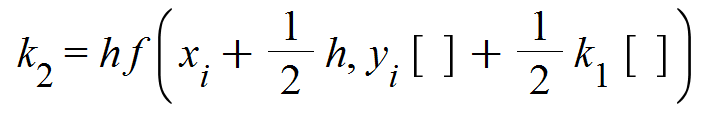
Before we go onto solving the two problems, we first implement a 4th-order Runge-Kutta (RK) method for an initial value problem for a system of N first-order ODEs in Maple.   
There are five input parameters for this procedure: (See line 1-5 of [1] in Maple Listing)

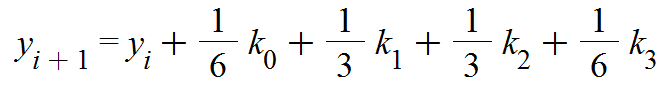
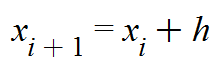
* *f* defines the vector functions for the method to use, as it is a vector function, it must return a list.
* *x0* defines the initial value of the independent variable.
* *xend* defines the end point for the method. (“end time” will refer to this variable in this report)
* *y0* is the list of the initial values of the dependent variables, in order word, it defines the initial conditions for the ODEs.
* Lastly, *n* defines the number of step the method should take which contribute to calculating the step size in the procedure.

In the 4th order RK procedure, first we calculate the step size h using the input parameters: *x0, xend, n*.

So we introduce variable h and define as (see line 8 of [1] in Maple Listing)

Next, we store *x0, y0* as *xi, yi* respectively and create two empty list (*tlist* and *wlist*) which will store the first and new values of these variables as we loop through the method. (see line 10-14 of [1] in Maple Listing)

Now, we will construct the loop for calculation of k0, k1, k2, k3, xi, yi for the 4th order Runge-Kutta.  
The loop would be a ‘for’ loop from 1 to number of step *n* (See line 15 of [1]).  
Within the ‘for’ loop, we will have the following equation:   
    
  

yi [ ] means a list of dependent variables at y(xi).

As we are dealing with a systems of N first-order ODEs, our *function f* will output a list of N values of the dependent variable and our initial conditions will be a list of N initial values of the dependent variables at 0. Therefore, it is necessary to define our k’s and y’s as a list of N integers in Maple calculation otherwise the values wouldn’t not compute in the procedure.

Note: yi+1 and xi+1 in the maple listing are written as yi and xi, as the procedure is reusing the variable but with the new value stored in the variable. (See line 16-21 of [1] in Maple Listing for the equations)

After we have computed new values of xi and yi, we push these variables into *tlist* and *wlist* respectively and continue the loop until *n* (see line 23-24 of [1]). At the end of the loop, *tlist* would have stored all the independent variable from *x0* to *xend*, and *wlist* would have stored all the list of yi values (list of dependent variables), creating a list of n lists with N (number of first-order ODEs) dependent variables within them.

Lastly, the procedure will return *tlist* and *wlist* as a single list where *tlist* is the first variable of the returned list and *wlist* is the second variable of the returned list (See line 31 of [1]).

The procedures use 10 digits in the calculation (See line 7 of [1]).

*Printing the results*

Within the procedures, there are various “printf” function which will print the step size *h* (See line 9 of [1]), the last independent variable *xend* (See line 27 of [1]), along with the dependent variable that goes with it *yi*. To print the dependent variable, we use a simple ‘for’ loop and print the each dependent variable in *yi* list from 1 to number element in *yi* list (See line 28-30 of [1]).  
Since this print ‘for’ loop coded is outside of the 4th order RK ‘for’ loop, the values in the yi will correspond to the last independent variable which is *xend*.

*Sorting the data points*

To plot the data obtained from the 4th Order Runge-Kutta procedure, we would need to separate the dependent variables from the data so they are stored individually,   
e.g. data = [ [t1,t2], [[x1,y1],[x2,y2]] ], separate them so xdata =[x1,x2], ydata = [y1,y2]   
then combine the independent variable with the dependent variables to obtain a list of dependent variable against independent variables.  
e.g. data[1] = [t1,t2], xdata = [x1,x2], ydata = [y1,y2] => txdata = [[t1,x1],[t2,x2]], tydata = [[t1,y1],[t2,y2]]

To separate the dependent variables, first we need to initialise empty data variables to store them, the number of empty data variable will be dependent on the system of ODEs (See line 1 of [3],[8], line 6-7 of [11], line 4-5 of [16], line 7-8 of [17]).

Next, we will use a ‘for’ loop from 1 to number of elements in list of dependent variable (denote as “nameofdata”[2] or *wlist* in the procedure) (See line 3 of [1],[8], line 11 of [11], line 8 of [16], line 11 of [17]), we extract the data and push them into the empty variables. (See line 4-5 of [3], line 4-6 of [8], line 7-12 of [16], and line 10-15 of [11],[17]).

Note that for [11],[16] and [17], we copied the list of dependent variable and used to the copied list to extract the data and push them into the empty variables. This will save computing time as we do not need to refer to the main data variable for every iteration in the ‘for’ loop.

Now, we set the newly extracted data into a list. (See line 7 of [1], line 9-11 of [8], line 16-21 of [16], and line 19-24 of [11],[17]).

To finish, we use the Maple function “zip”, to combine the independent variables with the extracted value which will give a list of data like txdata and tydata in the example above. (See line 6-9 of [3], line 12-14 of [8], line 22-27 of [16], line 25-30 of [11],[17]) .

This is necessary to use the “zip” several times to get all the dependent data with the independent data. We could also use the “zip” to combine two dependent data together (with displacement and velocity, See line 11 of [3]).

*Plotting graphs*

After obtaining these list of data points, we use the maple build-in plot function to plot:

* the angular displacement and the angular velocity against time (Question 1.b.i)
* the angular displacement against the angular velocity (Question 1.b.ii)
* x(t), y(t) and z(t) against time on the same graph (Question 2.b.i, 2.c.i, 2.d.i)

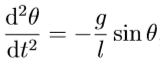
Using the maple pointplot3d function to plot the trajectory (x(t),y(t),z(t)) in the xyz-space, in the three-dimensional phase domain for Question 2.b.ii, 2.c.i, 2.c.ii.

(See [9],[10],[12],[13],[14],[15],[18] in Maple for the code of plotting the graph).

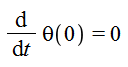
*Results and Discussion*

*Problem 1: Nonlinear motion of a simple pendulum*

The equation governing the motion of a pendulum given by the problem:

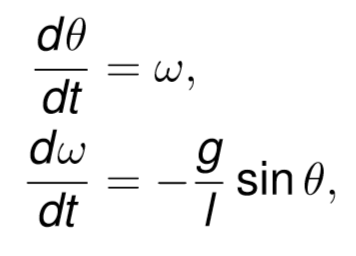
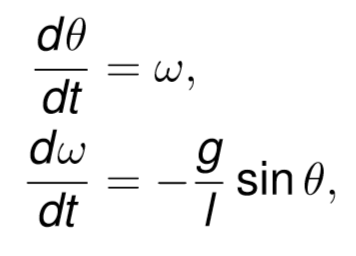
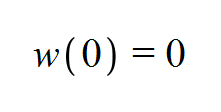


Where the initial conditions are:

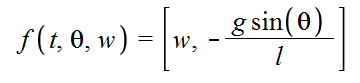
, , ,, .

* Alpha represent my personal constant. (See line 1 of [1]).
* g represent the acceleration of gravity in m/s2. (See line 2 of [1]).
* L represent the pendulum length. (See line 3 of [1]).
* θ(t) denotes the angular displacement of the pendulum.
* θ’(t) denotes the angular velocity of the pendulum.

The equation given by the problem is in 2nd order ODE. Since the 4th order Runge-Kutta procedure only takes system of first order ODEs, we rewrite the equation into 1st order ODEs.   
We obtain the following 1st order ODEs:

, where  and , where ..

And the equation we input into the procedure will be:



(See line 4 of [2] in Maple Listing for the exact Maple coding).

We define our x0 = 0 (start time), y0 = [Pi/3, 0] (Our initial conditions for θ and w), xend = 10 (end time) and varying number of step n.

Now, we need to choose a step size h so that our 4th-order Runge-Kutta method solution are accurate to 3 decimal places. We know that the global truncation error of the 4th-order Runge-Kutta methods is O(h4). This implies that halving the step size will divides the error by 24 or a factor of 16. Based on this information, we set h = 1/16 (0.0625 or n=160) for our initial guess for the step size.

To show that our initial guess will obtain a good approximation, we will generate a range of results with various step size to compares the respective values of θ(t) and θ’(t) (or w) at t = 10.

We obtain the following results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n (Number of steps) | h (Step Size) | t (Time) | θ(t) (the angular displacement of the pendulum) | θ’(t) (w, the angular velocity of the pendulum) |
| 10 | 1.00000 | 10 | 0.0037541125 | -0.0017177710 |
| 20 | 0.50000 | 10 | -0.7813506263 | -0.6809882586 |
| 40 | 0.25000 | 10 | -0.9056191814 | -1.0824008080 |
| 80 | 0.12500 | 10 | -0.9123192541 | -1.0888950130 |
| 100 | 0.10000 | 10 | -0.9125819892 | -1.0886233680 |
| 125 | 0.08000 | 10 | -0.9126888403 | -1.0884569990 |
| 160 | 0.06250 | 10 | -0.9127349900 | -1.0883647660 |
| 200 | 0.05000 | 10 | -0.9127510313 | -1.0883268060 |
| 250 | 0.04000 | 10 | -0.9127575712 | -1.0883095290 |
| 320 | 0.03125 | 10 | -0.9127603866 | -1.0883014300 |
| 400 | 0.02500 | 10 | -0.9127613780 | -1.0882983760 |
| 500 | 0.02000 | 10 | -0.9127617779 | -1.0882970330 |
| 800 | 0.01250 | 10 | -0.9127620146 | -1.0882962940 |
| 1000 | 0.01000 | 10 | -0.9127620405 | -1.0882961930 |
| 10000 | 0.00100 | 10 | -0.9127620420 | -1.0882960800 |
| 100000 | 0.00010 | 10 | -0.9127621284 | -1.0882964570 |

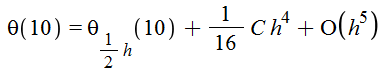
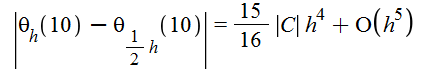
From the results, we can see that from our initial guess h=0.0625 onward will generate an approximation that is accurate up to 3 significant digits. So, the initial guess h=0.0625 manages to give a good approximation for the ODE.

Using the results and extrapolation techniques, we can work out a step size h to produce solutions that are accurate to 3 decimal places. Since the global truncation error is O(h4), we can write the truncation error E as .

If we take the step size to be h then we can write the exact value of θ(10) as follows:

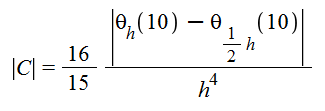
 (1)

Suppose we now take the step size to be h/2 then

 (2)

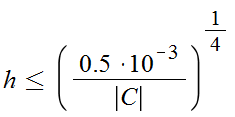
(1) - (2) =>

Rearranging this equation we get

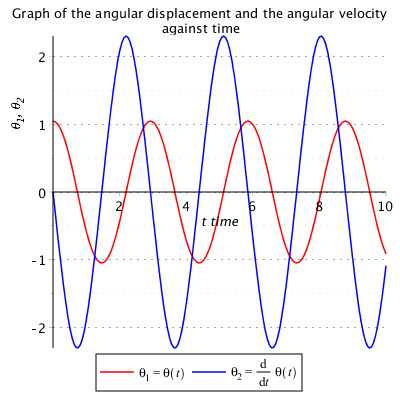


Now, we can estimate the constant. Using h = 0.02 and inputting the values from the results table, we get C = 1.750666667.

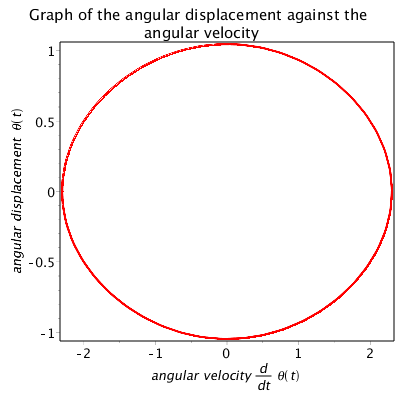
Using this constant and rearranging the error formula, we have

 implies 

0.0005 is to set the value of accuracy to 3 significant digit. Substitute C, we obtain h <= 0.1290436386. So for h smaller than this number, we will get an accurate approximation of the θ(10) up to 3 significate digit. Note, the values obtained from this method is just an estimate, 0.1290436386 is not exact.

Using the data obtained from using h=0.0625, we plot both the angular displacement θ(t) and the angular velocity θ’(t) against time, and the angular displacement θ(t) against the angular velocity θ’(t). (See [4] and [5] in Maple listing for the computation of plotting the graph)

As you can see from the graph on left, the red line represent the angular displacement and blue line represent the angular velocity. From this graph, we can see that both the angular displacement and velocity are oscillating.

This implies that the object is moving with a repeated pattern, at the same speed, acceleration and same distance repeatedly from time 0 to 10. This indicates the motion of a pendulum.

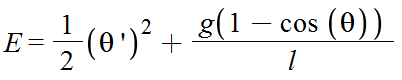
The graph on the right is the plot of the angular displacement θ(t) against the angular velocity θ’(t).   
As you can see on this graph, we obtain a circle. Starting with angular velocity = 0, following the line clockwise we can see that the displacement decreases to 0 (the minus indicate the direction of the displacement) while the velocity is increasing to around 2 (again the minus indicate the direction of the velocity).

This implies as the displacement decreases, the gravitational potential energy (PE) is converting to kinetic energy (KE) which explains the increase in velocity.

As we continue to observe the line, we see that while the displacement is increasing from 0 to 1, the velocity decreases to 0. This implies that the KE that we had before is converting back to PE as the displacement increases.

The whole process repeats again, and we obtain a circle. Since we can see a near perfect circle, this suggest that there is almost no loss of energy within this time period and there is a near perfect energy conversion between KE and PE.

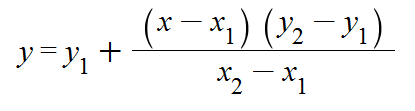
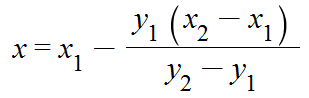
The equation for energy can be define by



Setting: t = 10, θ(t) = -0.9127349900, θ’(t) = -1.0883647660  
We get: E = 2.311777850

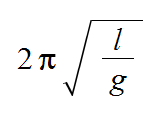
To find the period of oscillation of the pendulum, we need to find the time when the angular velocity reaches to 0 and multiply by 2 to obtain the period of the wave.

To calculate this in Maple, we first create a ‘for’ loop which will loop through the velocity data obtained from the 4th-Order Runge-Kutta and sorting the results (See line 1 of [6] for the loop, [3] for sorting the results). Next, using an ‘if’ statement, we compare whether the data in the iteration is less than 0 and the data in the next iteration is more than 0. (See line 2 of [6]). This is identify the two points which are nearest to zero, from positive and negative. If the ‘if’ statement is true, than we store the two velocity y0, y1 and time values, x0, x1. (See line 3-6 of [6]) Now, to estimate the time at velocity = 0, we will use linear interpolation.

Linear interpolation formula is as follows:  
   
Rearranging to subject of x with y = 0,  
 (See line 8 of [6]).

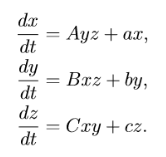
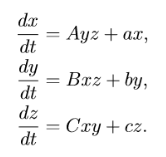
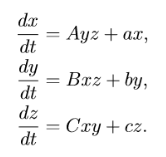
Now, we have worked out the time at velocity = 0, we multiply this result by 2 to obtain the period of the wave. (See line 9 of [6]).

Using our results we obtain, Period of Oscillation: 2.926733030.

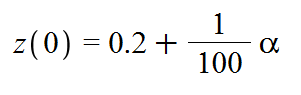
If we linearised the equation, we get  = 2.727139748.

We can see that the two solutions not quite the same but it is the data from the 4th-order Runge-Kutta is more accurate than the linearised solution because linearisation is of 1st order. However, through linearization, we can set sin(θ) ~= θ, this means we can solve the ODE analytically but a great deal of accuracy lost in return.

*Problem 2: Rigid Body Motion*

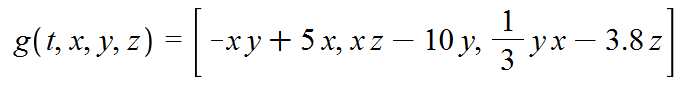
The following system of ordinary diﬀerential equations describes the motion of a rigid body when the applied torques depend linearly on the angular velocities about the principal axes of inertia ﬁxed at the centre of mass:  
   

Where A = -1, B = 1, C = 1/3, a = 5, b = -10 and c = -3.8   
and the initial conditions are:

, , , .  
up to time end = 2.

* Alpha represent my personal constant. (See line 1 of [1]).
* x(t), y(t) and z(t) denote the components of the angular velocity about the principal axes.
* A, B, C, a, b and c are reduced parameters

The equation we input into the procedure is:



(See line 2 of [7] for the equation).

We define our x0 = 0 (start time), y0 = [0.2, -0.2, 0.2+ α/100] (Our initial conditions for x, y, z), xend = 2 (end time) and varying number of step n.

To determine an optimal step size h, we input a range of different step size.

We obtain the following results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n (Number of steps) | h (Step Size) | t (Time) | X(t) | Y(t) | Z(t) |
| 20 | 0.10 | 2 | Undefined | Undefined | Undefined |
| 200 | 0.01 | 2 | 6. 1951575460 | 26. 2594963100 | 23. 7934906800 |
| 400 | 0.005 | 2 | 4. 0937385240 | 26. 0937385240 | 23. 8541623600 |
| 800 | 0.0025 | 2 | 4. 0077797730 | 26. 0174342000 | 23. 8542806200 |
| 1000 | 0.0020 | 2 | 4. 0054867300 | 26. 0170200100 | 23. 8542538600 |
| 2000 | 0.0010 | 2 | 4. 0042439890 | 26. 0167831100 | 23. 8542300100 |
| 2500 | 0.0008 | 2 | 4. 0042107930 | 26. 0167759700 | 23. 8542287100 |
| 4000 | 0.0005 | 2 | 4. 0041896860 | 26. 0167714500 | 23. 8542279800 |
| 8000 | 0.00025 | 2 | 4. 0041840520 | 26. 0167704300 | 23. 8542278500 |

Step Size that are larger than 0.02 are undefined due to numerical instability.

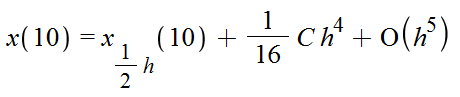
From the results, we can see that from h=0.0008 onward will generate an approximation that is accurate up to 4 significant digits.

Using the results and extrapolation techniques, we can work out a step size h to produce solutions that are accurate to 3 decimal places. Since the global truncation error is O(h4), we can write the truncation error E as .

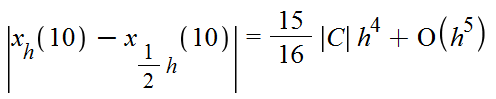
If we take the step size to be h then we can write the exact value of θ(10) as follows:

 (1)

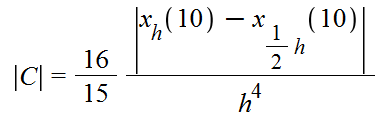
Suppose we now take the step size to be h/2 then

 (2)

(1) - (2) =>

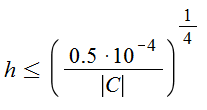


Rearranging this equation we get



Now, we can estimate the constant. Using h = 0.001 and inputting the values from the results table, we get C = 5.792320000x107

Using this constant and rearranging the error formula, we have

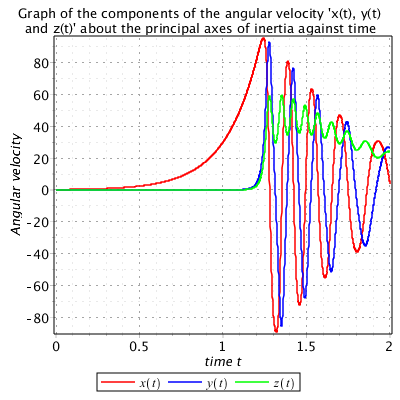
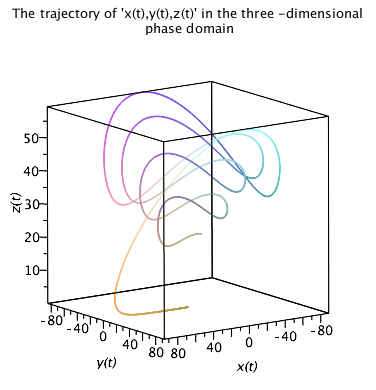
 implies 

0.00005 is to set the value of accuracy to 3 significant digit. Substitute C, we obtain h <= 0.0009638941944. So for h smaller than this number, we will get an accurate approximation of the x(10) up to 4 significant digit. Note, the values obtained from this method is just an estimate, 0.1290436386 is not exact.

Based on the results above and the extrapolation techniques, I believe the step size h = 0.0008 is the optional step size for this ODE.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n (Number of steps) | h (Step Size) | t (End Time) | X(t) | Y(t) | Z(t) |
| 2500 | 0.0008 | 2 | 4. 0042107930 | 26. 0167759700 | 23. 8542287100 |

Using the data obtained from using h=0.0008, we plot the components of the angular velocity x(t), y(t) and z(t) against time and the trajectory (x(t),y(t),z(t)) in the xyz-space, in the three-dimensional phase domain. For this three-dimensional phase domain, we use maple build-in pointplot3d function to plot the data (see [9] and [10] in Maple listing for the computation of plotting the graph)

As you can see from the graph on left, the red line represent x(t), blue line represent y(t) and green line represent z(t). From this graph, we can see that x(t) and y(t) displays a decaying oscillation starting from time = 1. The y(t)’s oscillation follows very closely to the x(t) with a different time period. It seems both of the lines are converging to zero but it is not confirmed since we only generated results up to 2 seconds.

While z(t) does somewhat show a decaying oscillation, however unlike x(t) and y(t), it oscillates only within the positive angular velocity. Following z(t), it started off with similar values to y(t) and started to oscillate at the same time as y(t). However, not only z(t) only oscillates within the positive angular velocity, it’s amplitude is very little compare to x(t) and y(t). There is indication that the line is converging to zero near the end of the graph but it is not confirmed due to insufficient results.

As you can see from the plot on the right, this is a representation of the x(t), y(t) and z(t) trajectories in three dimensional space. The spiral like line on the graph goes to a high z(t) and slowly circulating back to the bottom. This implies that z(t) go to a high velocity which explains the jump at the start, x(t) and y(t) are going from positive to negative and negative to positive which explains the spirals and the spirals are getting smaller and descending implies that all three trajectories are converging to zero.

For certain values of the parameters a, b and c, the model shows intriguing chaotic behaviour and can be used to illustrate the “butterﬂy eﬀect”. Now, we introduce two new initial conditions where

1. x(0) = 0.2, y(0) = −0.2 and z(0) = −0.2 + α/100 (See line 1 of [11] defined as para1)
2. ii. x(0) = 0.2, y(0) = 0.2 and z(0) = −0.2 (See line 2 of [11] defined as para2)

Using 4th-order Runge-Kutta, we compute the solutions up to time = 10.

We obtain the following results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n (Number of steps) | h (Step Size) | t (End Time) | x(t), y(t), z(t) of para1 | x(t), y(t), z(t) of para2 |
| 100 | 0.1 | 10 | undefined | undefined |
| 4000 | 0.0025 | 10 | 18.6709688300  -10.1184912900  -8.0888828070 | -7.1768151210 12.0502386600  -10.9542191400 |
| 8000 | 0.00125 | 10 | 18.8785446000  -9.6033481390  -7.6949910650 | -7.1816477210 12.0523436200  -10.9548623800 |
| 10000 | 0.00100 | 10 | 18.8823694600  -9.5918627220  -7.6862658890 | -7.1817814550 12.0524022400  -10.9548804100 |
| 20000 | 0.00050 | 10 | 18.8841202400  -9.5865801050  -7.6822530670 | -7.1818287460 12.0524224500  -10.9548862900 |
| 25000 | 0.00040 | 10 | 18.8839175300  -9.5871954210  -7.6827203230 | -7.1818513020 12.0524320300  -10.9548891100 |
| 40000 | 0.00025 | 10 | 18.8838085300  -9.5875279500  -7.6829726780 | -7.1818251860 12.0524209900  -10.9548859900 |
| 100000 | 0.00010 | 10 | 18.8836198900  -9.5881021630  -7.6834086210 | -7.1817695540 12.0523963600  -10.9548785100 |

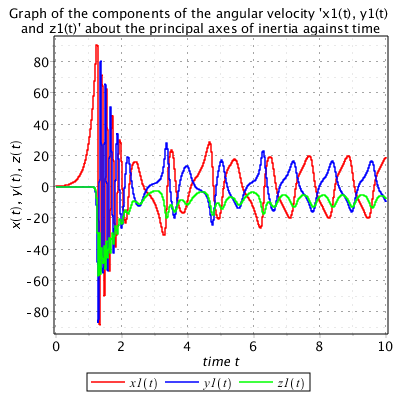
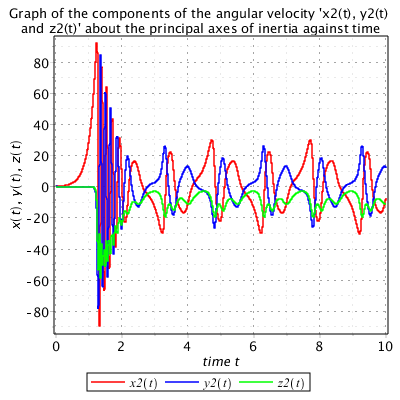
From the results, both initial conditions reaches an approximation that is accurate up to 3 significant digits around h = 0.00040. Using extrapolation techniques (see formulas above) and h = 0.0025, for para1 we obtain h <= 0.0005449807527, for para2 we obtain h <= 0.0007845671434.

So, the chosen step size will be h = 0.00025, the list of resulting values of x,y and z at t = 10 are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n (Number of steps) | h (Step Size) | t (End Time) | x(t), y(t), z(t) of para1 | x(t), y(t), z(t) of para2 |
| 25000 | 0.00040 | 10 | 18.8839175300  -9.5871954210  -7.6827203230 | -7.1818513020 12.0524320300  -10.9548891100 |

Using the data obtained from using h=0.00040, we plot the components of the angular velocity x(t), y(t) and z(t) against time and plot the trajectory (x(t),y(t),z(t)) in the xyz-space, in the three-dimensional phase domain for both of the initial conditions (para1 and para2).

Below shows the time phase for para1 and para2. (See [12],[13] in Maple listing for the computation of plotting the graph).

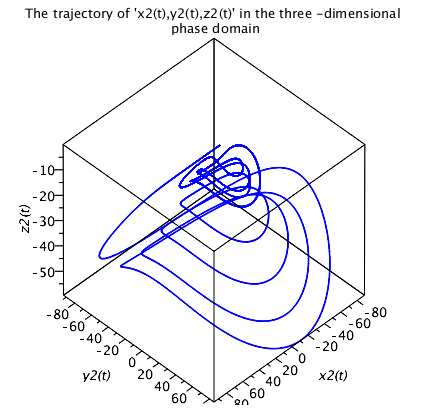
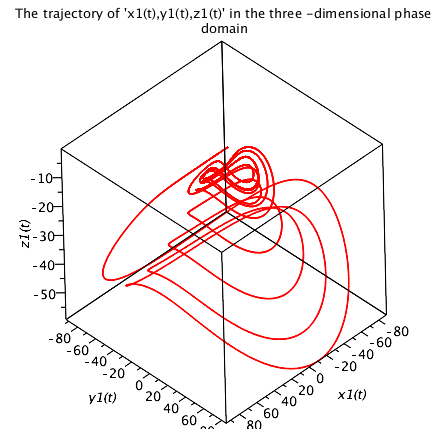
The graph on the left is plotted using the initial conditions in para1. The graph on the right is plotted using the initial conditions in para2. As you can see, at first both graph has similar oscillation with all three values x, y, z. However, the graph on the left has a different oscillation pattern compare to the right. On the right with para2, we can see a clear oscillation pattern from all x, y, z variables from time = 2 to time = 10. The oscillation repeats itself around t = 5 and t = 8 for x and y, z has a twice the frequency. However, with the left graph, after t = 7 onward, we can see the graph on the left started to generate different oscillation pattern. The oscillation pattern repeat itself every second, so it has a higher frequency then before. Comparing with the initial conditions are the beginning, the only difference was that para1 has z(0) = −0.2 + α/100 and para2 z(0) = −0.2 . With a difference of less than 0.005, the results obtained from this ODE has caused a major change in the oscillation. This behaviour is known as the butterfly effect.

In chaos theory, the butterfly effect is the sensitive dependence on initial conditions in which a small change in one state of a deterministic nonlinear system can result in large differences in a later state, rendering long-term prediction impossible in general. (Source [1]).

In simpler term, small changes in initial conditions can lead to great differences in outcome.

The example above has demonstrated this effect.

Below shows the 3d phase domain for para1 and para2. (See [14],[15] in Maple listing for the computation of plotting the graph).

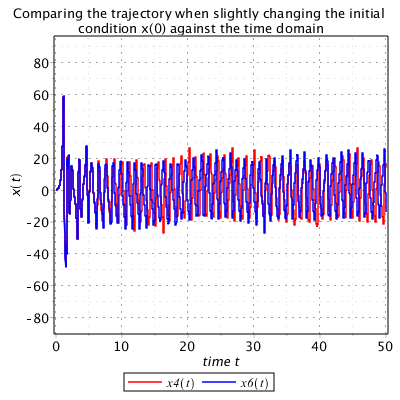
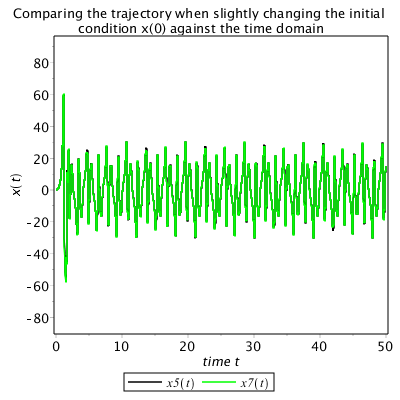


Comparing the two plots, we can see that there are more spirals in middle on the left graph comparing to right. This shows that as time increases, divergence started to appear for para1. Overall, these graphs implies that this system of ODEs has chaotic solutions.

Use the step size determined in i. and ii. above, we calculate the solutions for the same parameters for both sets of initial conditions and same step size but up to end time = 50. (See line 2-3 of [16] in Maple listing for the calculation).

Now, we introduce two new initial conditions. They both have the same initial conditions (parameters) as i. and ii. above but with x(0) = 0.2 + 10-3. So, we have

1. x(0) = 0.2 + 10-3, y(0) = −0.2 and z(0) = −0.2 + α/100 (See line 2 of [17] defined as para6)
2. x(0) = 0.2 + 10-3, y(0) = 0.2 and z(0) = −0.2 (See line 3 of [17] defined as para7)

We calculate the solutions with these new initial conditions with end time = 50 and compare the two trajectories x(t) by plotting them in the same plot (time domain). (See [17] in Maple listing for the computation of plotting the graph).   
 

The graph on the left is a plot of x(t) from para1 and para6. The graph on the right is a plot of x(t) from para2 and para7. As you can see from the graph on the right, the line of both x(t) are almost identical to each other. The graph on the left however, clearly shows the trajectory of the x(t)s are slightly different from each other. Looking at the initial conditions which generated this graph, the difference between the conditions from each graph are that z(0) = (-0.2 + α/100) on the left for both initial conditions and z(0) = -0.2 on the right for both initial conditions. Although the initial conditions x(t) has changed, we can see clearly from the graph one had shown the butterfly effect, while the other had not.

This implies that it is necessary to change the right dependent variable in order to observe the butterfly effect. The butterfly effect is the sensitive dependence on initial conditions in which a small change in one state of a deterministic nonlinear system can result in large differences in a later state. (Source [1]) However, changing the initial condition does not guarantee that we will obtain a chaotic solution, it is dependent on the variable you change.

The graph on the left also indicate that once we observed the butterfly effect, we could continue this effect by changing the other dependence variables, as long as the dependence variable that was responsible for the butterfly effect at the beginning is not reverted.

*Conclusion*

For a numerical methods to take in a system of 1st order differential equation, you need to make sure your input function outputs a list (or vector), your initial conditions are in a list (or vector) and the procedure must be able to perform mathematical operation on a list (or vector). For a 2nd-order ODEs, it is necessary to rewrite the equation as a system of 1st-order ODEs before using single step method such as 4th-order Runge-Kutta. The butterfly effect is the sensitive dependence on initial conditions but changing the initial condition will not guarantee a chaotic solution, it is dependent on the ODEs and the dependent variable you change.

*References*

Source [1] = <https://en.wikipedia.org/wiki/Butterfly_effect>

Source [2] = <https://en.wikipedia.org/wiki/Chaos_theory>