MAE 3210 - Spring 2019 - Project 3

Project 3 is due **online** through Canvas by 11:59PM on Thursday, April 25.

IMPORTANT REMARKS (Please read carefully):

- You are required to submit code for all functions and/or subroutines built to solve these problems, which is designed to be easy to read and understand, in your chosen programming language, **and which you have written yourself**. The text from your code should both be copied into a single PDF file submitted on canvas. **Your submitted PDF must also include responses to any assigned questions, which for problems requiring programming should be based on output from your code**. For example, if you are asked to find a numerical answer to a problem, the number itself should be included in your submission.
- In addition to what is required for homework submission, for each numerical answer you reach based on running code you have written yourself, **your submitted project PDF must include a copy of a screenshot** that shows your computer window after your code has been executed, with the numerical answer displayed clearly on the screen.

1. (BONUS POINTS ONLY: Worth up to 10 points, but not beyond 100% on the project)
   (a) Develop an algorithm which, for a given function of two variables $f(x, y)$, interval bounds a and b with $a < b$, and c and d with $c < d$, and input integer $n \geq 1$, does the following:
      (i) If n is odd, it applies the trapezoidal rule in each dimension to approximate $I = \int_c^d \left(\int_a^b f(x, y)\, dx \right) dy$.
      (ii) If n is even, it applies the multiple-application Simpson's 1/3 rule in each dimension to approximate $I = \int_c^d \left(\int_a^b f(x, y)\, dx \right) dy$.
   (b) Suppose the temperature T (°C) at a point (x, y) on a 16 m^2 rectangular heated plate is given by
   $$T(x, y) = x^2 - 3y^2 + xy + 72,$$
   where $-2 \leq x \leq 2$ and $0 \leq y \leq 4$ (here x and y are measured in meters about a reference point at $(0, 0)$). Determine the average temperature of the plate:

(i) Analytically, to obtain a true value.

(ii) Numerically, using the algorithm you developed in question 1(a) above, and plot the true percent relative error ϵ_t as a function of n for $1 \leq n \leq 5$. Provide some interpretation of the results.

2. Write code for two separate algorithms to implement (a) Euler's method and (b) the standard 4th order Runge-Kutta method, for solving a given first-order **one-dimensional** ODE. Design the code to solve the ODE over a prescribed interval with a prescribed step size, taking the initial condition at the left end point of the interval as an input variable.

3. The drag force F_d (N) exerted on a falling object can be modeled as proportional to the square of the objects downward velocity v (m/s), with a constant of proportionality c_d (kg/m).

   (a) Assume that a falling object has mass $m = 100$ (kg) with a drag coefficient of $c_d = 0.25$ kg/m, and let $g = 9.81$ (m/s^2) denote the constant downward acceleration due to gravity near the surface of the earth. Starting from Newton's second law, explain the derivation of the following ODE for the downward velocity $v = v(t)$ of the falling object:

$$\frac{dv}{dt} = 9.81 - 0.0025v^2.$$

   (b) Suppose that this same object is dropped from an initial height of $y_0 = 2$ km. Approximating $g = 9.81$ m/s^2, determine when the object hits the ground by solving the ODE you derived in question 3(a) using

   (i) Euler's method.

   (ii) the standard 4th order Runge-Kutta method.

4. Write code for two separate algorithms to implement (a) Euler's method and (b) the standard 4th order Runge-Kutta method, for solving a given first-order **two-dimensional** system of ODEs. Design the code to solve the system of ODEs over a prescribed interval with a prescribed step size.

5. The motion of a damped mass spring is described by the following ODE

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0, \tag{1}$$

   where x = displacement from equilibrium position (m), t = time (s), m = mass (kg), k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

(a) Rewrite the 2nd order ODE (1) as a two-dimensional system of first order ODEs for the displacement $x = x(t)$ and velocity $v = v(t)$ of the mass attached to the spring.

(b) Assume that the mass is $m = 10$ kg, the stiffness $k = 12$ N/m, the damping coefficient is $c = 3$ N·s/m, the initial velocity of the mass is zero ($v(0) = 0$), and the initial displacement is $x = 1$ m ($x(0) = 1$). Solve for the displacement and velocity of the mass over the time period $0 \leq t \leq 15$, and plot your results for the displacement $x = x(t)$,

  (i) using Euler's method with step size $h = 0.5$, and then with step size $h = 0.01$.

  (ii) using the standard 4th order Runge-Kutta method with step size $h = 0.5$, and then with step size $h = 0.01$.

(c) Assume that the mass is $m = 10$ kg, the stiffness $k = 12$ N/m, the damping coefficient is $c = 50$ N·s/m, the initial velocity of the mass is zero ($v(0) = 0$), and the initial displacement is $x = 1$ m ($x(0) = 1$). Solve for the displacement and velocity of the mass over the time period $0 \leq t \leq 15$, and plot your results for the displacement $x = x(t)$,

  (i) using Euler's method with step size $h = 0.5$, and then with step size $h = 0.01$.

  (ii) using the standard 4th order Runge-Kutta method with step size $h = 0.5$, and then with step size $h = 0.01$.

(d) (BONUS POINTS - but not beyond 100 percent on the project) Use analytical methods you have learned in the ODE pre/co-requisite to this course to find the analytic solution $x = x(t)$ to each of the problems posed in questions 5(b) and 5(c), and plot the analytic solution you obtain against your numerical solutions for comparison.

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 2

Write code for two separate algorithms to implement (a) Euler's method and(b) the
standard 4th order Runge-Kutta method, for solving a given first-orderone-dimensional ODE.
Design the code to solve the ODE over a prescribed interval with a prescribed step size,
taking the initial condition at the left endpoint of the interval as an input variable.

@author: Jacob Needham
"""

def Main():
    #user input to define variables
    x0 =       #inital x value
    xF =       #final x value
    y0 =       #boundary condition
    h  =       #step size

    #RK
    #returning vectors for variable 1, variable 2, and x values
    RK_y1, x_values = RK4_2D(x0,y10,y20,xF,h)

    #Euler
    #returning vectors for variable 1, variable 2, and x values
    euler_y1, x_values = eulerMethod(x0,xF,y10,y20,h)


#differential function
def f1(x,y2):
    y = FUNCTION
    return y

#Runge-Kutte function for integrating based on a boundary condition.
def RK4_1D (x0,y10,y20,xF,h):

    #Creating vector holding all x variables to be iterated on
    x_values = np.arange(x0,xF+h,h)

    #initilizing solution vectors
    y1 = [0 for i in range(len(x_values))]
    y1[0] = y10

    #main RK4 loop
    for i in range(len(x_values)-1):
        K11 = f1(x_values[i],y1[i])

        K21 = f1(x_values[i]+.5*h,y1[i]+.5*K11*h)

        K31 = f1(x_values[i]+.5*h,y1[i]+.5*K21*h)

        K41 = f1(x_values[i]+h,y1[i]+K31*h)

        y1[i+1] = (y1[i]+(1/6)*(K11+2*K21+2*K31+K41)*h)

    return y1, x_values
```

```
def eulerMethod (x0,xF,y10,y20,h):
    x_values = np.arange(x0,xF+h,h)
    y_euler = [0 for i in range(len(x_values))]

    y_euler1[0] = y10

    for i in range(len(x_values)-1):
        y_euler[i+1] = y_euler[i] + f1(x_values[i],y_euler[i])*h

    return y_euler, x_values
```

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 3

The drag force Fd (N) exerted on a falling object can be modeled as proportional to the
square of the objects downward velocity v (m/s), with a constant of proportionality
cd (kg/m).
(a) Assume that a falling object has mass m = 100 (kg) with a drag coefficient of
cd = 0.25 kg/m, and let g = 9.81 (m/s2) denote the constant downward acceleration due to
gravity near the surface of the earth. Starting from Newton's second law, explain the
derivation of the following ODE for the downward velocity v = v(t) of the falling object:
dv/dt = 9.81 − 0.0025v^2
(b) Suppose that this same object is dropped from an initial height of y0 = 2 km.
Determine when the object hits the ground by solving the ODE you derived in question 3(a)
using:
(i) Euler's method.
(ii) the standard 4th order Runge-Kutta method.

@author: Jacob Needham
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0        #initial time
    xF = 37       #final time guess
    y10 = 0       #intiial velocity
    y20 = 2000    #initial y position
    h1 = .01      #step size

    #RK
    #retrns velocity vector, position vector, and time vector
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta falling body problem h=.01')

    print()
    print()

    #Euler
    #retrns velocity vector, position vector, and time vector
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  falling body problem h=.01')


#differential velocity function
def f1(t,v,x):
    y = 9.81 - 0.0025*v**2
    return y


#differential position function
def f2(t,v,x):
    y = -v
    return y
```

```
def RK4_2D (x0,y10,y20,xF,h):

    #Creating vector holding all x variables to be iterated on
    x_values = np.arange(x0,xF+h,h)

    #initilizing velocity and position solution vectors
    y1 = [0 for i in range(len(x_values))]
    y2 = [0 for i in range(len(x_values))]
    y1[0] = y10
    y2[0] = y20

    #main for loop to RK4 the function
    for i in range(len(x_values)-1):
        K11 = f1(x_values[i],y1[i],y2[i])
        K12 = f2(x_values[i],y1[i],y2[i])

        K21 = f1(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)
        K22 = f2(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)

        K31 = f1(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)
        K32 = f2(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)

        K41 = f1(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)
        K42 = f2(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)

        y1[i+1] = (y1[i]+(1/6)*(K11+2*K21+2*K31+K41)*h)
        y2[i+1] = (y2[i]+(1/6)*(K12+2*K22+2*K32+K42)*h)

        #testing to see of the position is less than 0, or hit the ground
        if y2[i+1] <= 0:
            print('Using RK-4, the time at which the object ' \
                  'hits the ground is:',x_values[i],'seconds.')
            break

    return y1, y2, x_values

def eulerMethod (x0,xF,y10,y20,h):

   #Creating vector holding all x variables to be iterated on
    x_values = np.arange(x0,xF+h,h)

    #initilizing velocity and position solution vectors
    y_euler1 = [0 for i in range(len(x_values))]
    y_euler2 = [0 for i in range(len(x_values))]
    y_euler1[0] = y10
    y_euler2[0] = y20

    #main Euler function
    for i in range(len(x_values)-1):
        y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
        y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h

        #testing to see of the position is less than 0, or hit the ground
        if y_euler2[i+1] <= 0:
            print('Using Euler\'s Method, the time at which the object ' \
                  'hits the ground is:',x_values[i],'seconds.')
```

```
            break

    return y_euler1, y_euler2, x_values


#simple function to graph axis
def function(x):
    y = x
    return y

#fFunction to graph
def plotSpace (timeSeries,yVelocity,yPosition,title):

    #setting up graph
    graph.xlabel('Time (s)')
    graph.ylabel('Velocity (m/s) or Postion (m)')
    graph.title('Project 3 Problem 3 \n' + title)

    #graphing velocity
    graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
    graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')

    #graphing postion
    graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
    graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')

    #showing legend
    graph.legend()

    #plotting axis
    x = np.arange(min(timeSeries),max(timeSeries),.01)
    y = function(x)
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()


Main()
```

Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 3.py

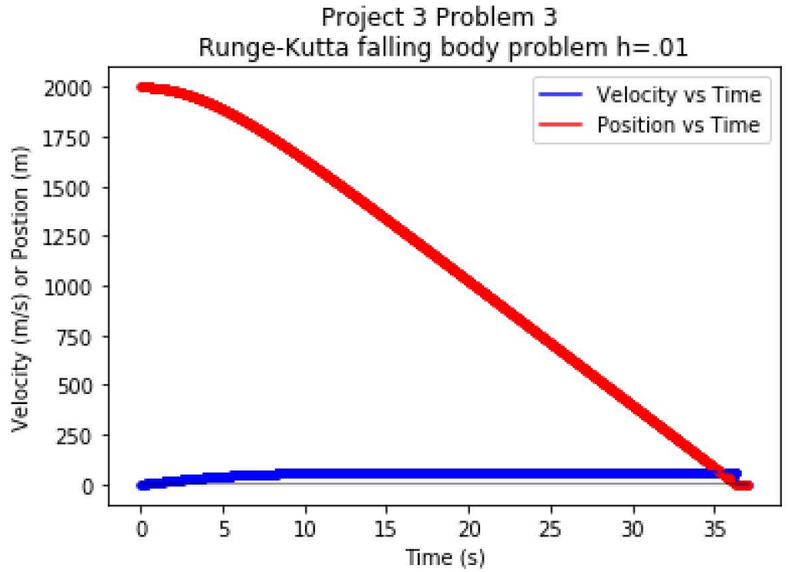
Problem 2.py | Problem 3.py | Problem 4 Part B.py | Problem 4 Part C.py

```
                'hits the ground is:',x_values[i],'seconds.')
            break

    return y_euler1, y_euler2, x_values


#simple function to graph axis
def function(x):
    y = x
    return y

#fFunction to graph
def plotSpace (timeSeries,yVelocity,yPosition,title):

    #setting up graph
    graph.xlabel('Time (s)')
    graph.ylabel('Velocity (m/s) or Postion (m)')
    graph.title('Project 3 Problem 3 \n' + title)

    #graphing velocity
    graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
    graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')

    #graphing postion
    graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
    graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')

    #showing legend
    graph.legend()

    #plotting axis
    x = np.arange(min(timeSeries),max(timeSeries),.01)
    y = function(x)
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()


Main()
```

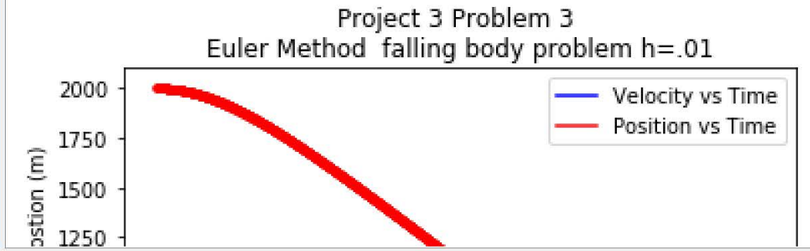
IPython console

Console 5/A

```
In [30]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 3/Problem 3.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 3')
Using RK-4, the time at which the object hits the ground is: 36.35 seconds.
```




```
Using Euler's Method, the time at which the object hits the ground is: 36.35 seconds.
```



Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 130 | Column: 51 | Memory: 49 %

Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 3.py

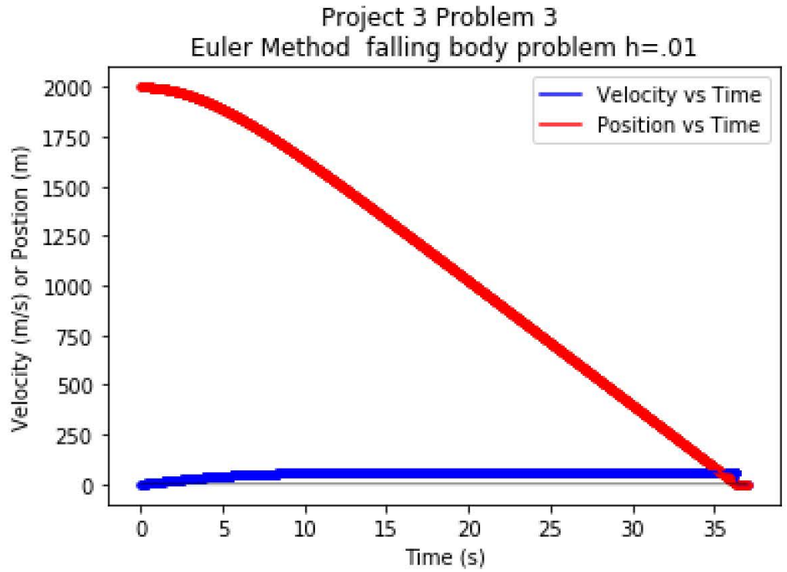
Problem 2.py | Problem 3.py | Problem 4 Part B.py | Problem 4 Part C.py

```
                'hits the ground is:',x_values[i],'seconds.')
            break

    return y_euler1, y_euler2, x_values


#simple function to graph axis
def function(x):
    y = x
    return y

#fFunction to graph
def plotSpace (timeSeries,yVelocity,yPosition,title):

    #setting up graph
    graph.xlabel('Time (s)')
    graph.ylabel('Velocity (m/s) or Postion (m)')
    graph.title('Project 3 Problem 3 \n' + title)

    #graphing velocity
    graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
    graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')

    #graphing postion
    graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
    graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')

    #showing legend
    graph.legend()

    #plotting axis
    x = np.arange(min(timeSeries),max(timeSeries),.01)
    y = function(x)
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()


Main()
```

IPython console

Console 5/A

```
Using Euler's Method, the time at which the object hits the ground is: 36.35
seconds.
```




```
In [31]:
```

New file | Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 130 | Column: 51 | Memory: 41 %

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 4
Write code for two separate algorithms to implement (a) Euler's method and (b) the
standard 4th order Runge-Kutta method, for solving a given first-order two-dimensional
system of ODEs.  Design the code to solve the system of ODEs over a prescribed interval
with a prescribed step size.

@author: Jacob Needham
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0  =        #inital x value
    xF  =        #final x value
    y10 =        #boundary condition for variable 1
    y20 =        #boundary condition for variable 2
    h1  =        #step size

    #RK
    #returns velocity vectors, position vector, possible x locations vector
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)

    #Euler
    #returns velocity vectors, position vector, possible x locations vector
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)


#differential velocity function
def f1(t,v,x):
    y = FUNCTION
    return y


#differential position funciton
def f2(t,v,x):
    y = FUNCTION
    return y


def RK4_2D (x0,y10,y20,xF,h):

    #this section creates an array with all x value spacing by h
    x_values = np.arange(x0,xF+h,h)

    #initilizing solution vectors
    y1 = [0 for i in range(len(x_values))]
    y2 = [0 for i in range(len(x_values))]
    y1[0] = y10
    y2[0] = y20

    #main for loop to RK4 the function
    for i in range(len(x_values)-1): #end case is not include, in matlab it would be len-1
        K11 = f1(x_values[i],y1[i],y2[i])
```

```
        K12 = f2(x_values[i],y1[i],y2[i])

        K21 = f1(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)
        K22 = f2(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)

        K31 = f1(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)
        K32 = f2(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)

        K41 = f1(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)
        K42 = f2(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)

        y1[i+1] = (y1[i]+(1/6)*(K11+2*K21+2*K31+K41)*h)
        y2[i+1] = (y2[i]+(1/6)*(K12+2*K22+2*K32+K42)*h)

    return y1, y2, x_values


def eulerMethod (x0,xF,y10,y20,h):
    x_values = np.arange(x0,xF+h,h)
    y_euler1 = [0 for i in range(len(x_values))]
    y_euler2 = [0 for i in range(len(x_values))]

    y_euler1[0] = y10
    y_euler2[0] = y20

    for i in range(len(x_values)-1):
        y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
        y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h

    return y_euler1, y_euler2, x_values

Main()
```

Project 3 Problem 5 (A)

Rewrite the 2nd Order ODE (2) as a two dimensional system of the first order ODEs for the displacement $x = x(t)$ and velocity $v = v(t)$ of the mass attached to the spring

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

realize that $\frac{d^2x}{dt^2} = \frac{dv}{dt}$ and $\frac{dx}{dt} = v$

$$\therefore m\frac{dv}{dt} + cv + kx = 0$$

$$\frac{dv}{dt} = \frac{-cv + kx}{m} \quad (1)$$

and

$$\frac{dy}{dt} = v \quad (2)$$

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 4 part B
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part B:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient
is c = 3 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and
then with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=3, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=3, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
    plotSpace(x_values, RK1_h2, RK2_h2, 'Runge-Kutta  c=3, m=10, k=12, h=.01')

    #Plot of Euler for c=3 and h=1
    euler1_h2, euler2_h2, x_values = eulerMethod(x0,xF,y10,y20,h2)
    plotSpace(x_values, euler1_h2, euler2_h2, 'Euler Method  c=3, m=10, k=12, h=.01')

#funciton really is dv/dt
def f1(t,v,x):
    c = 3
    k = 12
    m = 10
    y = -((c/m)*v)-((k/m)*x)
    return y
```

```
#function really is d2y/dt2
def f2(t,v,x):
    y = v
    return y


def RK4_2D (x0,y10,y20,xF,h):

    #this section creates an array with all x value spacing by h
    x_values = np.arange(x0,xF+h,h)

    #initilizing solution vectors
    y1 = [0 for i in range(len(x_values))]
    y2 = [0 for i in range(len(x_values))]
    y1[0] = y10
    y2[0] = y20

    #main for loop to RK4 the function
    for i in range(len(x_values)-1):
        K11 = f1(x_values[i],y1[i],y2[i])
        K12 = f2(x_values[i],y1[i],y2[i])

        K21 = f1(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)
        K22 = f2(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)

        K31 = f1(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)
        K32 = f2(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)

        K41 = f1(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)
        K42 = f2(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)

        y1[i+1] = (y1[i]+(1/6)*(K11+2*K21+2*K31+K41)*h)
        y2[i+1] = (y2[i]+(1/6)*(K12+2*K22+2*K32+K42)*h)

    return y1, y2, x_values

def eulerMethod (x0,xF,y10,y20,h):
    x_values = np.arange(x0,xF+h,h)
    y_euler1 = [0 for i in range(len(x_values))]
    y_euler2 = [0 for i in range(len(x_values))]

    y_euler1[0] = y10
    y_euler2[0] = y20

    for i in range(len(x_values)-1):
        y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
        y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h

    return y_euler1, y_euler2, x_values



#Function to graph
def plotSpace (timeSeries,yVelocity,yPosition,title):

    #setting up graph
    graph.xlabel('Time (s)')
```

```
    graph.ylabel('Velocity (m/s) or Postion (m)')
    graph.title('Project 3 Problem 5 \n' + title)

    #graphing velocity
    graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
    graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')

    #graphing postion
    graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
    graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')

    #showing legend
    graph.legend()

    #plotting axis
    x = np.arange(min(timeSeries),max(timeSeries),.01)
    y = function(x)
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()


Main()
```

Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part B.py

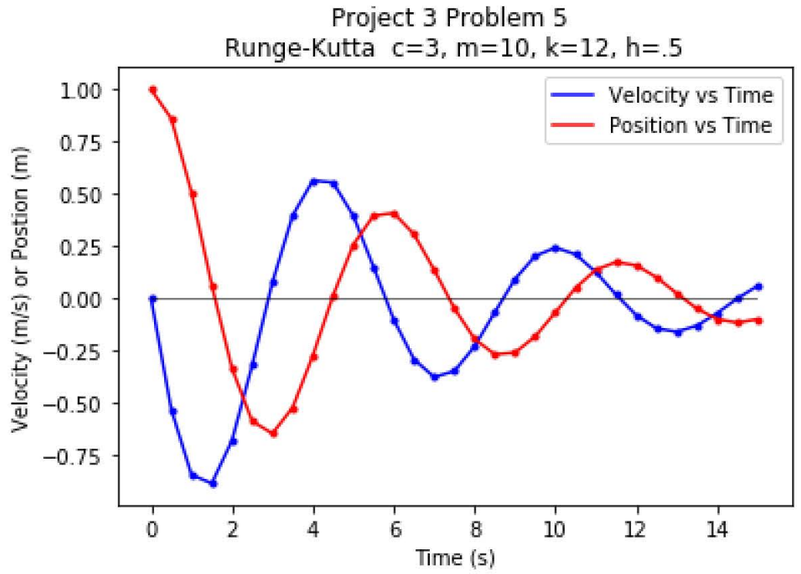
Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

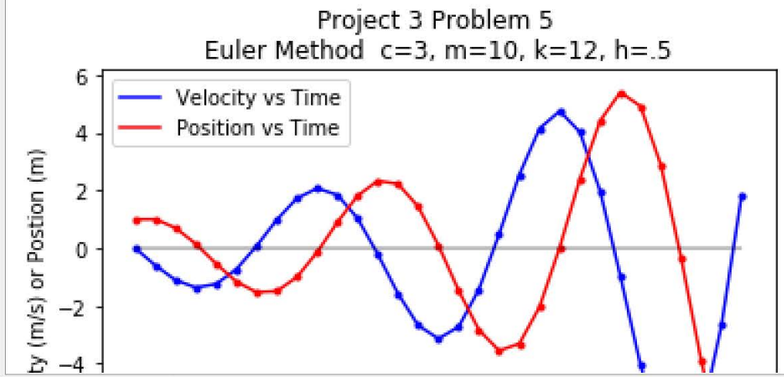
```
86 def eulerMethod (x0,xF,y10,y20,h):
87     x_values = np.arange(x0,xF+h,h)
88     y_euler1 = [0 for i in range(len(x_values))]
89     y_euler2 = [0 for i in range(len(x_values))]
90 
91     y_euler1[0] = y10
92     y_euler2[0] = y20
93 
94     for i in range(len(x_values)-1):
95         y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
96         y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h
97 
98     return y_euler1, y_euler2, x_values
99 
100 
101 
102 #Function to graph
103 def plotSpace (timeSeries,yVelocity,yPosition,title):
104 
105     #setting up graph
106     graph.xlabel('Time (s)')
107     graph.ylabel('Velocity (m/s) or Postion (m)')
108     graph.title('Project 3 Problem 5 \n' + title)
109 
110     #graphing velocity
111     graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
112     graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')
113 
114     #graphing postion
115     graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
116     graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')
117 
118     #showing legend
119     graph.legend()
120 
121     #plotting axis
122     x = np.arange(min(timeSeries),max(timeSeries),.01)
123     y = function(x)
124     graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
125     #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')
126 
127     #grpahing
    graph.show()
```

IPython console

Console 5/A

```
In [34]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 3/Problem 5 Part B.py', wdir='C:/Users/Needh/Documents/Spring 2019/Numerical Methods/Project 3')
```





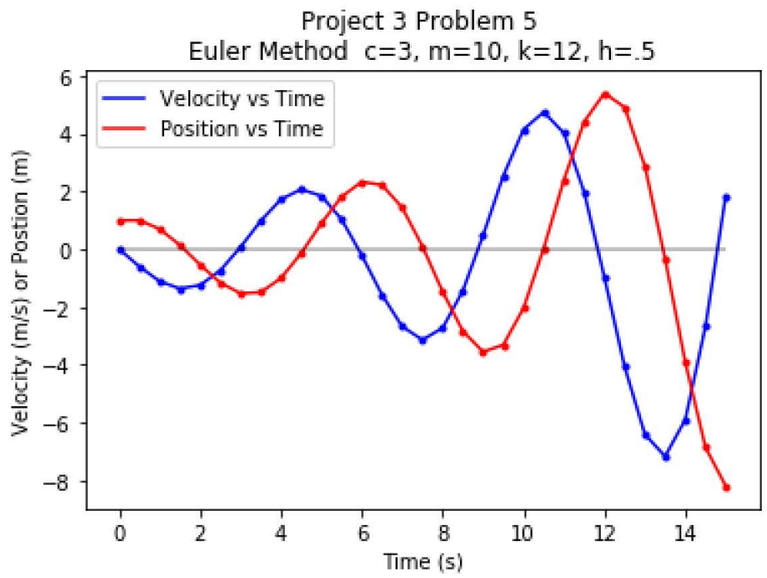
Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part B.py

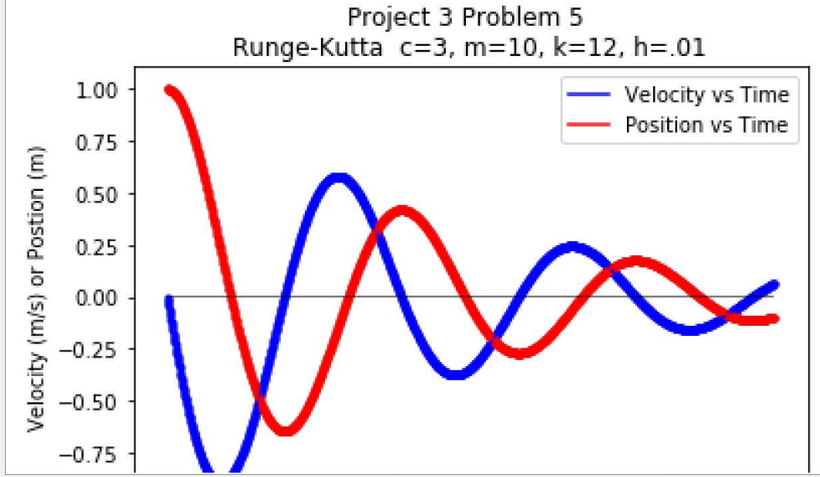
Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```
86 def eulerMethod (x0,xF,y10,y20,h):
87     x_values = np.arange(x0,xF+h,h)
88     y_euler1 = [0 for i in range(len(x_values))]
89     y_euler2 = [0 for i in range(len(x_values))]
90 
91     y_euler1[0] = y10
92     y_euler2[0] = y20
93 
94     for i in range(len(x_values)-1):
95         y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
96         y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h
97 
98     return y_euler1, y_euler2, x_values
99 
100 
101 
102 #Function to graph
103 def plotSpace (timeSeries,yVelocity,yPosition,title):
104 
105     #setting up graph
106     graph.xlabel('Time (s)')
107     graph.ylabel('Velocity (m/s) or Postion (m)')
108     graph.title('Project 3 Problem 5 \n' + title)
109 
110     #graphing velocity
111     graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
112     graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')
113 
114     #graphing postion
115     graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
116     graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')
117 
118     #showing legend
119     graph.legend()
120 
121     #plotting axis
122     x = np.arange(min(timeSeries),max(timeSeries),.01)
123     y = function(x)
124     graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
125     #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')
126 
127     #grpahing
    graph.show()
```

IPython console

Console 5/A

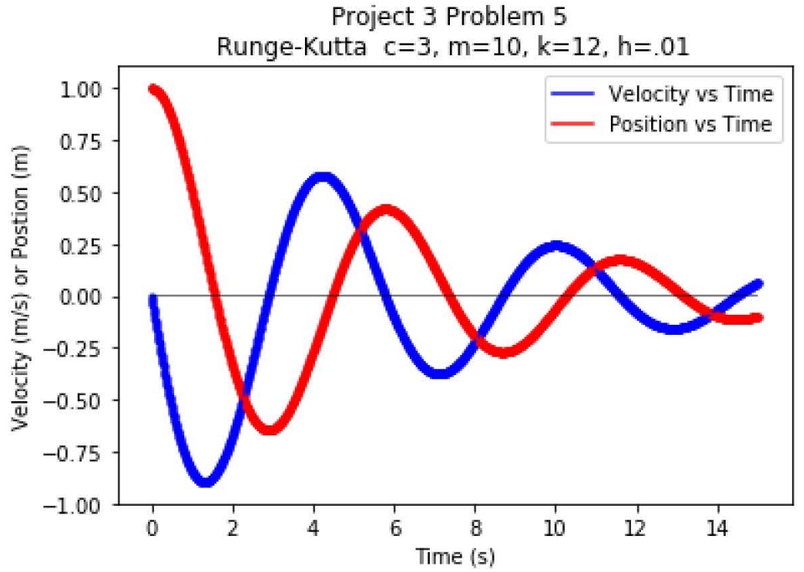


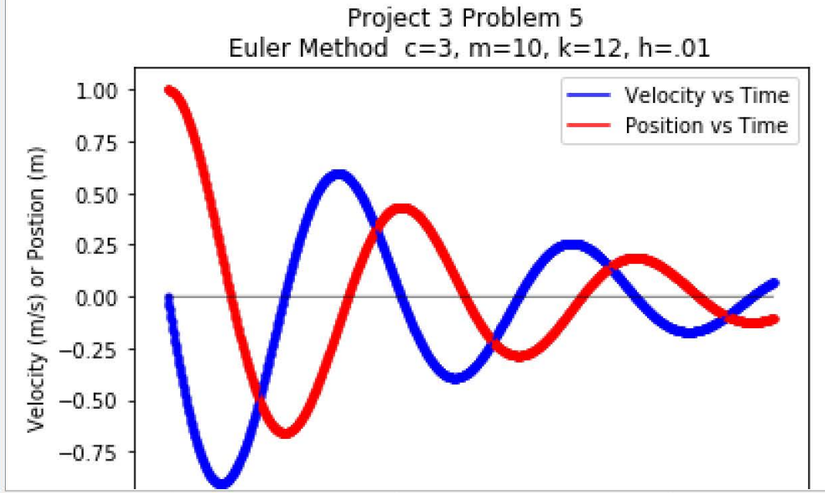


Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```
86 def eulerMethod (x0,xF,y10,y20,h):
87     x_values = np.arange(x0,xF+h,h)
88     y_euler1 = [0 for i in range(len(x_values))]
89     y_euler2 = [0 for i in range(len(x_values))]
90 
91     y_euler1[0] = y10
92     y_euler2[0] = y20
93 
94     for i in range(len(x_values)-1):
95         y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
96         y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h
97 
98     return y_euler1, y_euler2, x_values
99 
100 
101 
102 #Function to graph
103 def plotSpace (timeSeries,yVelocity,yPosition,title):
104 
105     #setting up graph
106     graph.xlabel('Time (s)')
107     graph.ylabel('Velocity (m/s) or Postion (m)')
108     graph.title('Project 3 Problem 5 \n' + title)
109 
110     #graphing velocity
111     graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
112     graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')
113 
114     #graphing postion
115     graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
116     graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')
117 
118     #showing legend
119     graph.legend()
120 
121     #plotting axis
122     x = np.arange(min(timeSeries),max(timeSeries),.01)
123     y = function(x)
124     graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
125     #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')
126 
127     #grpahing
    graph.show()
```

Console 5/A





Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **108** Column: **37** Memory: **41 %**

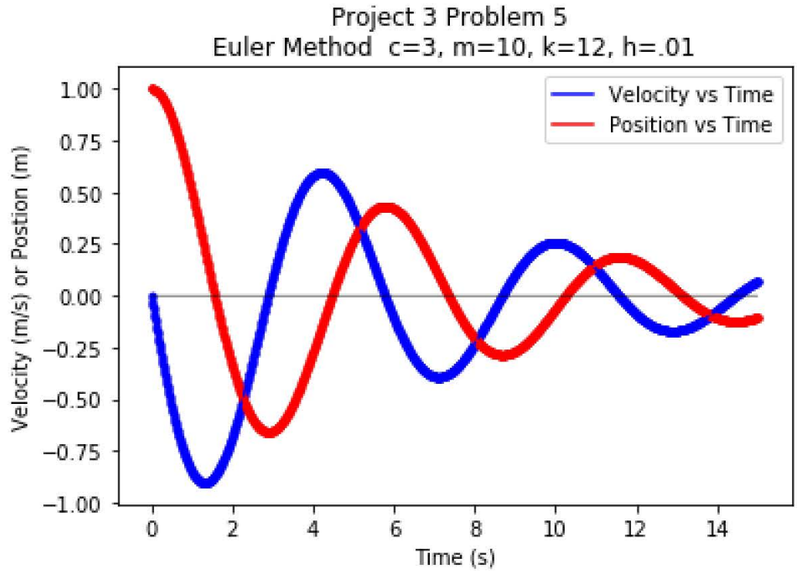
Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part B.py

Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```
86 def eulerMethod (x0,xF,y10,y20,h):
87     x_values = np.arange(x0,xF+h,h)
88     y_euler1 = [0 for i in range(len(x_values))]
89     y_euler2 = [0 for i in range(len(x_values))]
90 
91     y_euler1[0] = y10
92     y_euler2[0] = y20
93 
94     for i in range(len(x_values)-1):
95         y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
96         y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h
97 
98     return y_euler1, y_euler2, x_values
99 
100 
101 
102 #Function to graph
103 def plotSpace (timeSeries,yVelocity,yPosition,title):
104 
105     #setting up graph
106     graph.xlabel('Time (s)')
107     graph.ylabel('Velocity (m/s) or Postion (m)')
108     graph.title('Project 3 Problem 5 \n' + title)
109 
110     #graphing velocity
111     graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
112     graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')
113 
114     #graphing postion
115     graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
116     graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')
117 
118     #showing legend
119     graph.legend()
120 
121     #plotting axis
122     x = np.arange(min(timeSeries),max(timeSeries),.01)
123     y = function(x)
124     graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
125     #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')
126 
127     #grpahing
    graph.show()
```

IPython console

Console 5/A



In [35]:

New file | Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 108 | Column: 37 | Memory: 41 %

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 5 part C
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part C:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient is
c = 50 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and then
with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=50, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=50, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
    plotSpace(x_values, RK1_h2, RK2_h2, 'Runge-Kutta  c=50, m=10, k=12, h=.01')

    #Plot of Euler for c=3 and h=1
    euler1_h2, euler2_h2, x_values = eulerMethod(x0,xF,y10,y20,h2)
    plotSpace(x_values, euler1_h2, euler2_h2, 'Euler Method  c=50, m=10, k=12, h=.01')


#funciton really is dv/dt
def f1(t,v,x):
    c = 50
    k = 12
    m = 10
    y = -((c/m)*v)-((k/m)*x)
    return y
```

```
#function really is d2y/dt2
def f2(t,v,x):
    y = v
    return y


def RK4_2D (x0,y10,y20,xF,h):
    #this section creates an array with all x value spacing by h
    x_values = np.arange(x0,xF+h,h)
    #initilizing solution vectors
    y1 = [0 for i in range(len(x_values))]
    y2 = [0 for i in range(len(x_values))]
    y1[0] = y10
    y2[0] = y20

    #main for loop to RK4 the function
    for i in range(len(x_values)-1):
        K11 = f1(x_values[i],y1[i],y2[i])
        K12 = f2(x_values[i],y1[i],y2[i])

        K21 = f1(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)
        K22 = f2(x_values[i]+.5*h,y1[i]+.5*K11*h,y2[i]+.5*K12*h)

        K31 = f1(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)
        K32 = f2(x_values[i]+.5*h,y1[i]+.5*K21*h,y2[i]+.5*K22*h)

        K41 = f1(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)
        K42 = f2(x_values[i]+h,y1[i]+K31*h,y2[i]+K32*h)

        y1[i+1] = (y1[i]+(1/6)*(K11+2*K21+2*K31+K41)*h)
        y2[i+1] = (y2[i]+(1/6)*(K12+2*K22+2*K32+K42)*h)

    return y1, y2, x_values

def eulerMethod (x0,xF,y10,y20,h):
    x_values = np.arange(x0,xF+h,h)
    y_euler1 = [0 for i in range(len(x_values))]
    y_euler2 = [0 for i in range(len(x_values))]

    y_euler1[0] = y10
    y_euler2[0] = y20

    for i in range(len(x_values)-1):
        y_euler1[i+1] = y_euler1[i] + f1(x_values[i],y_euler1[i],y_euler2[i])*h
        y_euler2[i+1] = y_euler2[i] + f2(x_values[i],y_euler1[i],y_euler2[i])*h

    return y_euler1, y_euler2, x_values



#Function to graph
def plotSpace (timeSeries,yVelocity,yPosition,title):

    #setting up graph
    graph.xlabel('Time (s)')
    graph.ylabel('Velocity (m/s) or Postion (m)')
```

```
    graph.title('Project 3 Problem 5 \n' + title)

    #graphing velocity
    graph.scatter(timeSeries,yVelocity, color = 'blue', marker = '.')
    graph.plot(timeSeries,yVelocity, color = 'blue', label = 'Velocity vs Time')

    #graphing postion
    graph.scatter(timeSeries,yPosition, color = 'red', marker = '.')
    graph.plot(timeSeries,yPosition, color = 'red', label = 'Position vs Time')

    #showing legend
    graph.legend()

    #plotting axis
    x = np.arange(min(timeSeries),max(timeSeries),.01)
    y = function(x)
    graph.plot(x, x*0 + 0 , linewidth = .5, color = 'black')
    #graph.plot(x*0 +0, y, linewidth =.5, color = 'black')

    #grpahing
    graph.show()


Main()
```

Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part C.py

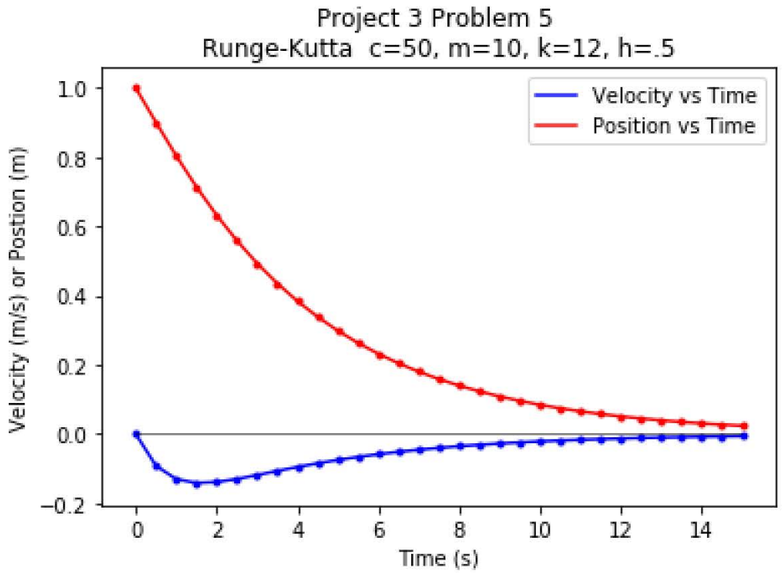
Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

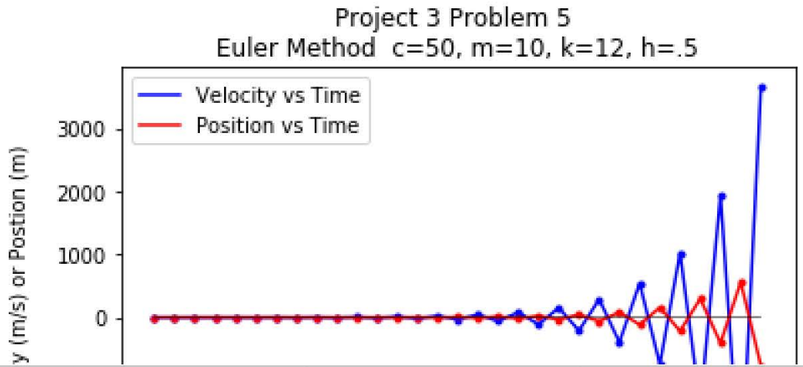
```python
# -*- coding: utf-8 -*-
"""
Project 3 Problem 5 part C
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part C:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient is
c = 50 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and then
with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=50, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=50, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
```

IPython console

Console 5/A

```
In [36]: runfile('C:/Users/Needh/Documents/Spring 2019/Numerical Methods/
Project 3/Problem 5 Part C.py', wdir='C:/Users/Needh/Documents/Spring
2019/Numerical Methods/Project 3')
```





Permissions: **RW** | End-of-lines: **CRLF** | Encoding: **UTF-8** | Line: **62** | Column: **13** | Memory: **41 %**

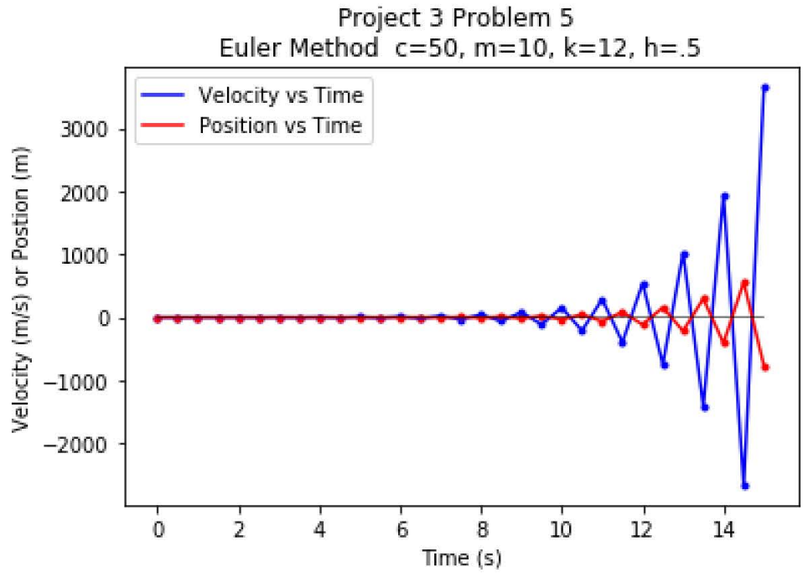
Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part C.py

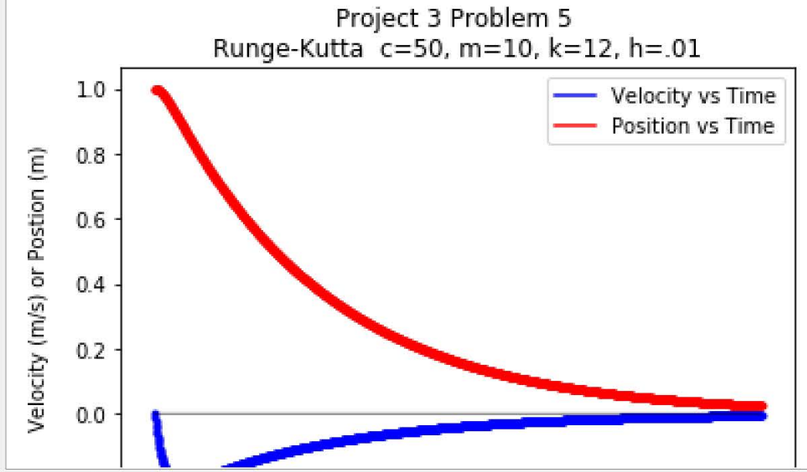
Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```python
# -*- coding: utf-8 -*-
"""
Project 3 Problem 5 part C
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part C:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient is
c = 50 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and then
with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=50, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=50, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
```

IPython console

Console 5/A





New file | Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 62 | Column: 13 | Memory: 41 %

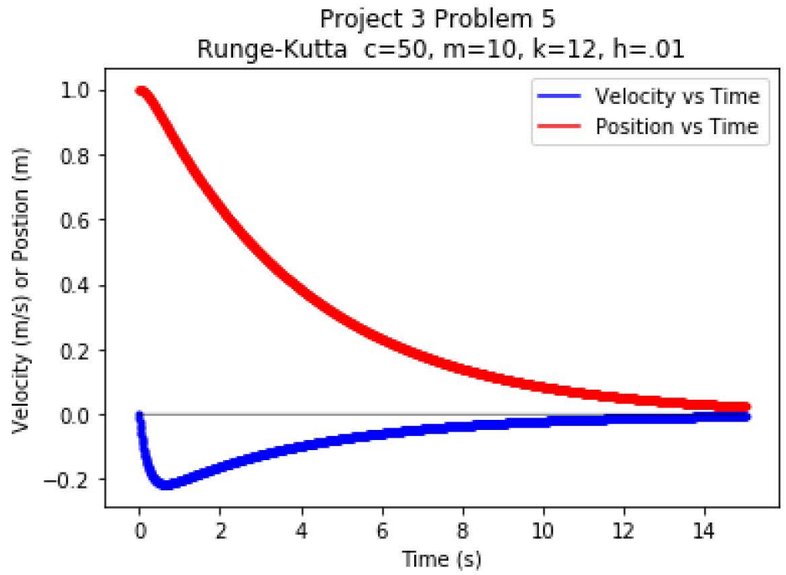
Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part C.py

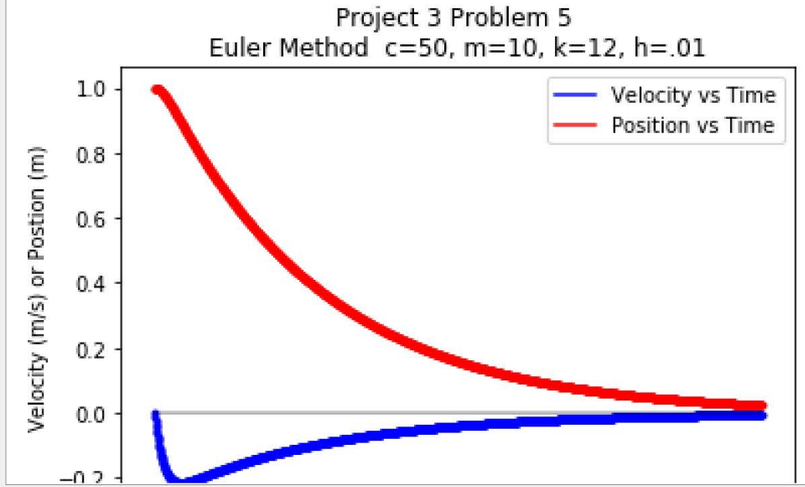
Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```python
# -*- coding: utf-8 -*-
"""
Project 3 Problem 5 part C
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part C:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient is
c = 50 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and then
with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=50, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=50, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
```

IPython console

Console 5/A





New file | Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 62 | Column: 13 | Memory: 40 %

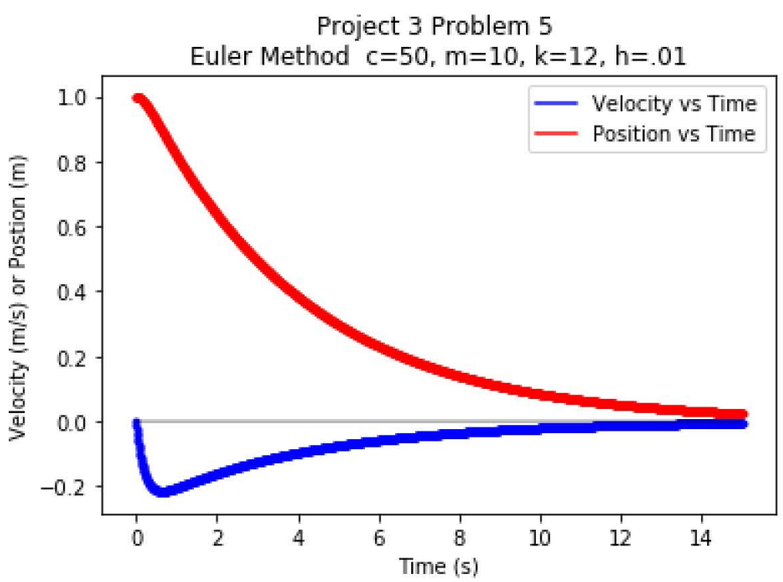
Editor - C:\Users\Needh\Documents\Spring 2019\Numerical Methods\Project 3\Problem 5 Part C.py

Problem 2.py | Problem 3.py | Problem 4.py | Problem 5 Part B.py | Problem 5 Part C.py

```
# -*- coding: utf-8 -*-
"""
Project 3 Problem 5 part C
The motion of a damped mass spring is described by the following ODE :
m * d2x/dt2 + c * dx/dt + kx = 0,
where x = displacement from equilibrium position (m), t = time (s), m = mass (kg),
k = stiffness constant (N/m) and c = damping coefficient (N·s/m).

Part C:
Assume that the mass is m = 10 kg, the stiffness k = 12 N/m, the damping coefficient is
c = 50 N·s/m, the initial velocity of the mass is zero (v(0) = 0), and the initial
displacement is x = 1 m (x(0) = 1). Solve for the displacement and velocity of the mass
over the time period 0 ≤ t ≤ 15, and plot your results for the displacement x = x(t),
(i) using Euler's method with step size h = 0.5, and then with step size h = 0.01.
(ii) using the standard 4th order Runge-Kutta method with step size h = 0.5, and then
with step size h = 0.01.

@author: Jacob Neehdam
"""

import matplotlib.pyplot as graph
import numpy as np

def Main():
    #input perameters from user
    x0 = 0
    xF = 15
    y10 = 0
    y20 = 1
    h1 = .5
    h2 = .01

    #plot of RK for c=3 and h=.5
    RK1_h1, RK2_h1, x_values = RK4_2D(x0,y10,y20,xF,h1)
    plotSpace(x_values, RK1_h1, RK2_h1, 'Runge-Kutta  c=50, m=10, k=12, h=.5')

    #Plot of Euler for c=3 and h=.5
    euler1_h1, euler2_h1, x_values = eulerMethod(x0,xF,y10,y20,h1)
    plotSpace(x_values, euler1_h1, euler2_h1, 'Euler Method  c=50, m=10, k=12, h=.5')

    #Plot of RK for c=3 and h=1
    RK1_h2, RK2_h2, x_values = RK4_2D(x0,y10,y20,xF,h2)
```

IPython console

Console 5/A



In [37]:

Permissions: RW | End-of-lines: CRLF | Encoding: UTF-8 | Line: 62 | Column: 13 | Memory: 40 %