Professor : If the code can't copy properly, please visit to download code or check properly.

➔ https://github.com/mecitpmk/Numerical-Methods-Problems-Solutions

Problem 1.2

Attention : This question written by **Python.** Therefore, If you want to run code correctly please be sure **numpy** and **matplotlib** installed as library.

```
import numpy as np
def EstimatingVelocity(givenStepSize):
    """
    givenStepSize > a Number that interval  givenStepSize > 0

    """
    lastVelocity =  0
    initalTime = 0
    gravity = 9.81
    velocities = {}
    velocities[initalTime] = 0
    for step in np.arange(givenStepSize,9,givenStepSize):
        if (step > 8):
            break
        lastVelocity = lastVelocity + (gravity - ((12.5/68.1)
*lastVelocity))*givenStepSize
        velocities[step] = lastVelocity

    return velocities


import matplotlib.pyplot as plt
velocitiesFirst = EstimatingVelocity(1) # for step size is 1
velocitiesSecond = EstimatingVelocity(0.5) # for step size 05
figure1 , axs  = plt.subplots(2)
figure1.suptitle('Graph of Velocities')
axs[0].plot(list(velocitiesFirst.keys()),list(velocitiesFirst
.values()),label='Step for 1')
axs[0].legend()
axs[0].grid(True)
axs[1].plot(list(velocitiesSecond.keys()),list(velocitiesSeco
nd.values()),label='Step for 0.5',color='red')
axs[1].legend()
```

```
axs[1].grid(True)
plt.xlabel("Time(t)")
plt.ylabel("Velocity(m/s)")
plt.show()
```

➔ See the graph in **Figure 1.1**

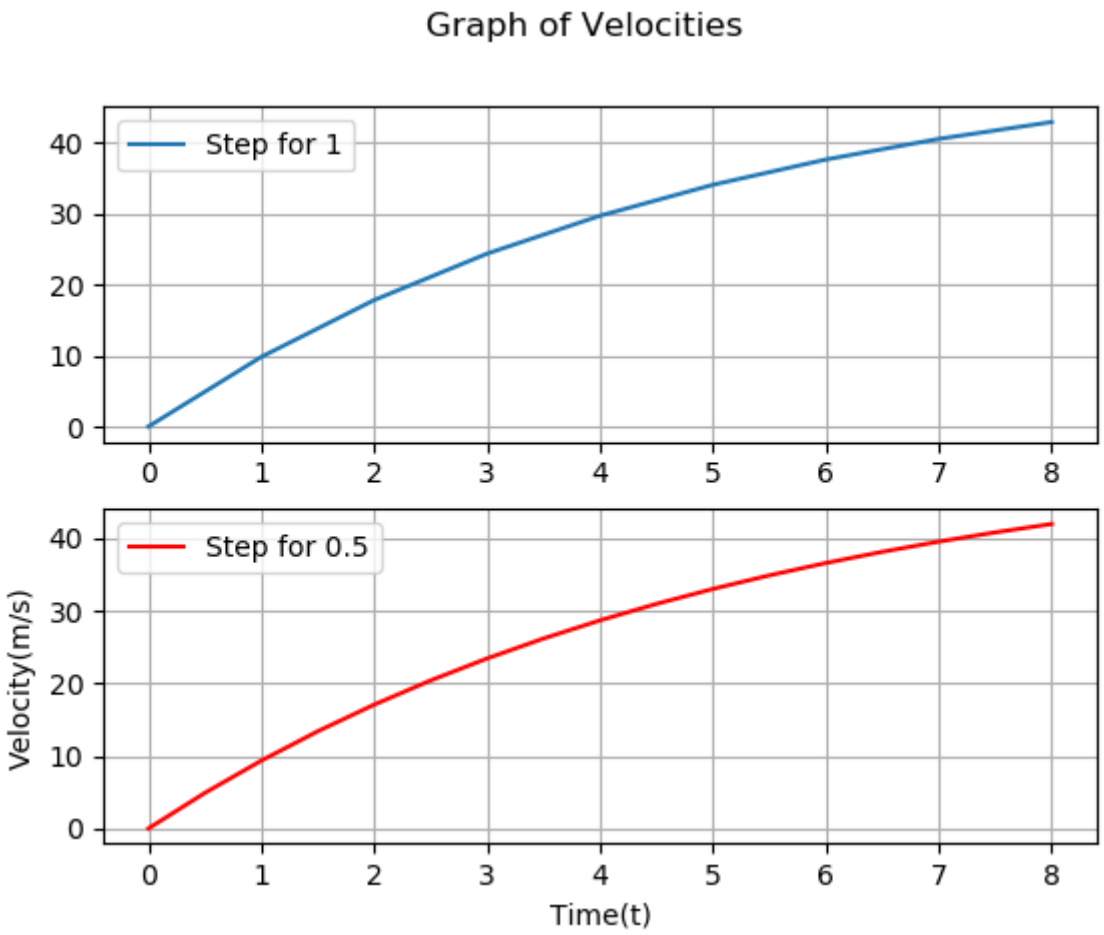

Figure 1.1

Result : Error is small is the step size is small.

Problem 1.9

Attention : This question written by **Python.** Therefore, If you want to run code correctly please be sure **numpy** and **matplotlib** and **math** installed as library.

```
import matplotlib.pyplot as plt
import numpy as np
import math
def tankProblem(tStart = 0 , tEnd = 10 , tStep = 0.5):
    A = 1250
    Q = 450
    CalculatedY = 0
    tAndYvalues = {}
    tAndYvalues[tStart] = 0
    for steps in np.arange(tStart+tStep, tEnd , tStep):
        interval = steps-tStep
        CalculatedY = CalculatedY + ((3*Q/A*math.pow(math.sin
(interval),2))-Q/A)*tStep
        tAndYvalues[steps] = CalculatedY
    return tAndYvalues

tankProblems = tankProblem()
print(tankProblems)
plt.plot(list(tankProblems.keys()),list(tankProblems.values()
),label="Water Level")
plt.xlabel("Time(t)")
plt.ylabel("Level(m)")
plt.legend()
plt.grid(True)
plt.show()
```

➔ See the graph in **Figure 2.2**

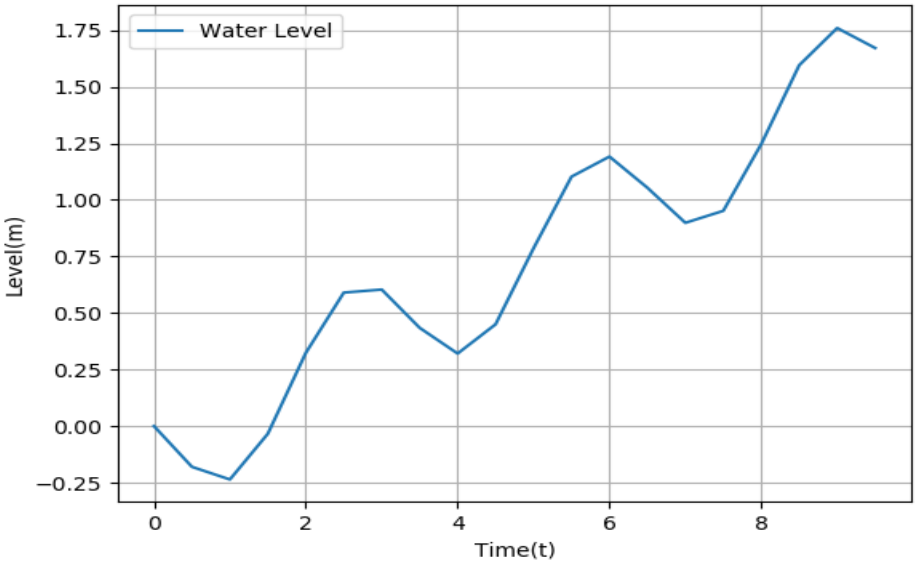
Water Level
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
Level(m)
0
2
4
6
8
Time(t)

Figure 2.2

Problem 1.15

Attention : This question written by **Python.** Therefore, If you want to run code correctly please be sure **numpy** and **matplotlib** installed as library.

```
import matplotlib.pyplot as plt
import numpy as np
def forCircuit(): # Q 1.15
    forT = [i for i in np.arange(0,0.11,0.01)]
    i = [0]
    q = [1]
    for varK in range(1,11):
        i.append(i[varK-1]-(i[varK-1]*200+q[varK-
1]/0.0001)/5 *0.01)
        q.append(q[varK-1]+i[varK-1]*0.01)
    return forT , i , q

t , i , q  = forCircuit()
plt.suptitle('I and Q versus T')
plt.plot(t,i,label='I')
plt.plot(t,q,label='Q')
plt.legend()
plt.xlabel("Time(t)")
plt.grid(True)
plt.show()
```

See the graph in **Figure 3.3**

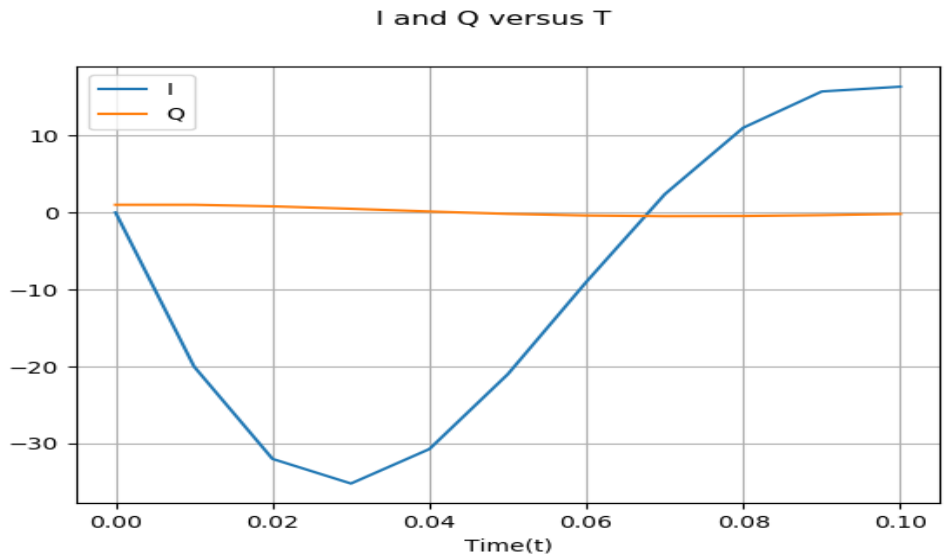

Figure 3.3

Problem 3.5

Attention: This question written by **Matlab** because of the floating point is much more precise than python.

```
function finiteChecker(startFrom,endFrom)
    incrementType = 0;
    if (startFrom > endFrom)
        incrementType = -1;
    else
        incrementType = 1;
    end
    s=0;
    for i=startFrom:incrementType:endFrom
        s=s+1/i^4;
    end
    fprintf('Value is : %.20f \n',s);
    errorCalculation = ((pi^4)/90 -s)/((pi^4)/90);
    fprintf('Error :');
    disp(errorCalculation);
end
```

Hint : When you try to run this code, please don't forget to call function in command window. See in **Figure 4.4**

- ➔ finiteChecker(1,10000)
- ➔ finiteChecker(10000,1)

```
>> finiteChecker(1,10000)
Value is : 1.08232323371086103236
Error :   2.5583e-13


>> finiteChecker(10000,1)
Value is : 1.08232323371080485508
Error :   3.0773e-13
```

Figure 4.4

Result : The first approach yields a result closer to the sum limit, but the limit is not the exact value of the sum of the first 10,000 terms. On the contrary calculating the sum in reverse order yields a more accurate result.

Problem 3.6

Attention: This question written by **Matlab** because of the floating point is much more precise than python.

```
function twoApproaches(x,trueValue)
    term1=1;
    term2=1;
    sum1o=term1;
    sum2o=term2;
    for i=2: 20
        sum1=sum1o+(-1)^(i-1)*x^(i-1)/factorial(i-1);
        term2=term2+x^(i-1)/factorial(i-1);
        sum2=1/term2;
        et1=abs((trueValue-sum1)/sum1);
        et2=abs((trueValue-sum2)/sum2);
        ea1=abs((sum1-sum1o)/sum1);
        ea2=abs((sum2-sum2o)/sum2);
        sum1o=sum1;
        sum2o=sum2;
    end
    my_list = [sum1,et1,ea1,sum2,et2,ea2];
    fthEquation = 'First';
    scthEquation = 'Second';
    currentEquation = fthEquation;
    for i=1:3:length(my_list)
        if (i<4)
            currentEquation = fthEquation;
        else
            currentEquation = scthEquation;
        end
         fprintf('Summation for %s Eq : %s \n',currentEquatio
n,num2str(my_list(i)));
         fprintf('Et for %s Eq: %s \n',currentEquation,num2st
r(my_list(i+1)));
         fprintf('Error Approx. for %s Eq %s \n\n',currentEqu
ation,num2str(my_list(i+2)));
    end
end
```

➔ See the command window Result in **Figure 5.5**

```
>> twoApproaches(5,6.737947*10^-3)
Summation for First Eq : 0.0067063
Et for First Eq: 0.0047128
Error Approx. for First Eq 0.02338

Summation for Second Eq : 0.0067379
Et for Second Eq: 3.4508e-07
Error Approx. for Second Eq 1.0565e-06
```

Figure 5.5

Result: We see that the both methods converge towards the true value of e^{-5}, but the first one demonstrate large oscillations at the beginning and slower convergence.

Professor : If the code can't copy properly, please visit to download code or check properly.

➔ https://github.com/mecitpmk/Numerical-Methods-Problems-Solutions