Python Assignment - 3

121901050

Kunal Keshav Damame

Question 1

1. Solve the second order differential equation in Section 1.1 analytically for $f(x) = -1$, $g = 0$, and $h = 0$.

Analytical solution

1) $\dfrac{d^2\phi}{dx^2} = f(x) \qquad x \in [0, 1] \qquad \phi(0) = 0$

$$\left.\frac{d\phi}{dx}\right|_{x=1} = 0$$

$$\frac{d^2\phi}{dx^2} = -1$$

$$\left.\frac{d\phi}{dx}\right|_{x=1} = -x + c$$

$$0 = -1 + c \Rightarrow c = 1$$

$$\phi = \frac{-x^2}{2} + cx + d$$

$$\phi(0) = 0 \Rightarrow \qquad d = 0$$

$$\phi = -\frac{x^2}{2} + x.$$

Question 2

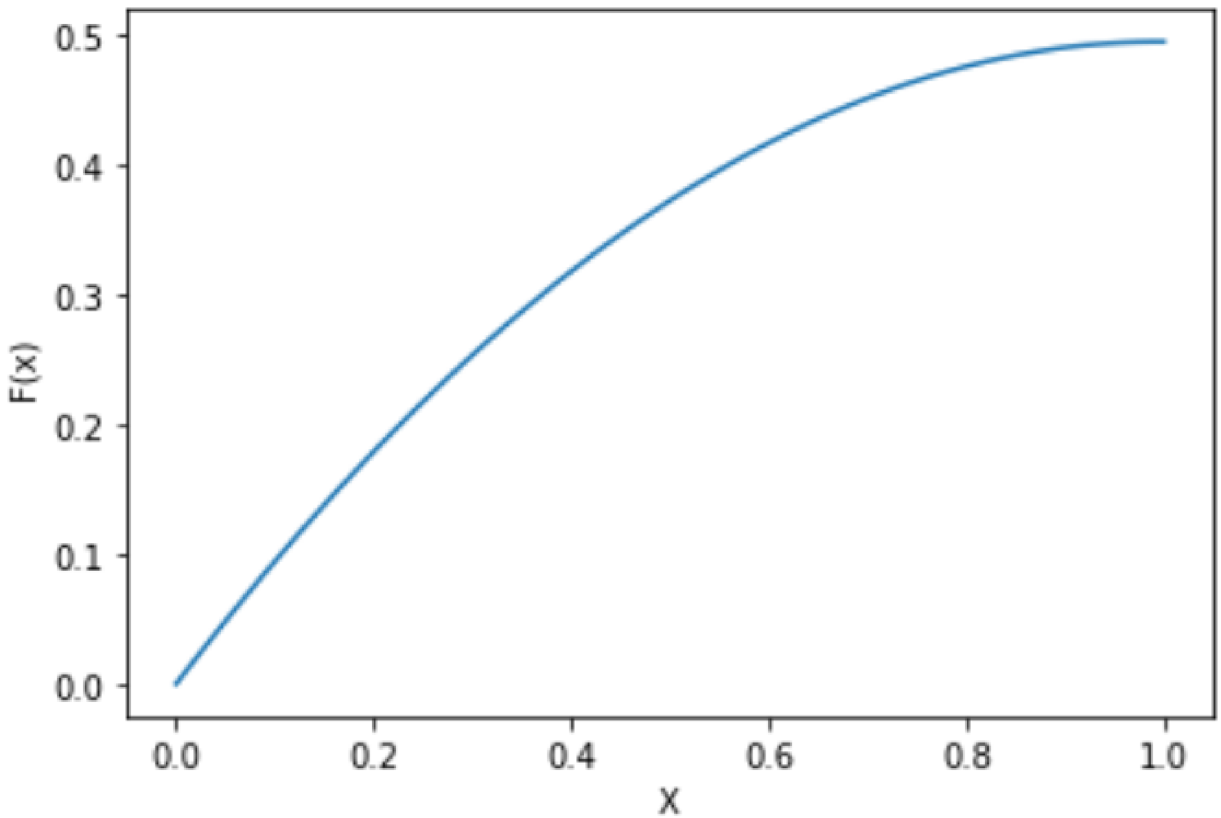
2. Write a Python program to solve the differential equation via the finite differences method.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

#specify the number of points
N = 100

#make the data points
divisions = np.linspace(0,1,N+1)

#value of delta will be the difference divided by N
Δ = 1/N

#initilize the array with the zeros
A = np.zeros((N+1,N+1))

#put the values of the boundary
A[0,0] = 1
A[N,N] = 1

#due the differential form of boundary condition
A[N,N-1] = -1

#make the matrix for the equations
#then assign the matrix according to the theroy
for i in range(1,N):
    A[i][i] = -2
    A[i][i-1] = 1
    A[i][i+1] = 1


#solutions of the x at each discraetized point
B = np.zeros(N+1)
B[1:-1] = -1*Δ*Δ

#solve the equation
```

```
y = np.linalg.solve(A,B)

#plot the graph
plt.plot(divisions,y)
plt.xlabel("X")
plt.ylabel("F(x)")
plt.show()
```

Graph:



Question 3:

3. Define error $e(N)$ as a function of discretization points as

$$e(N) = \frac{1}{N+1}\sum_{j=0}^{N} \frac{|\Phi_{\text{numerical}}(x) - \Phi_{\text{analytical}(x)}|}{|\Phi_{\text{analytical}(x)}|}.$$

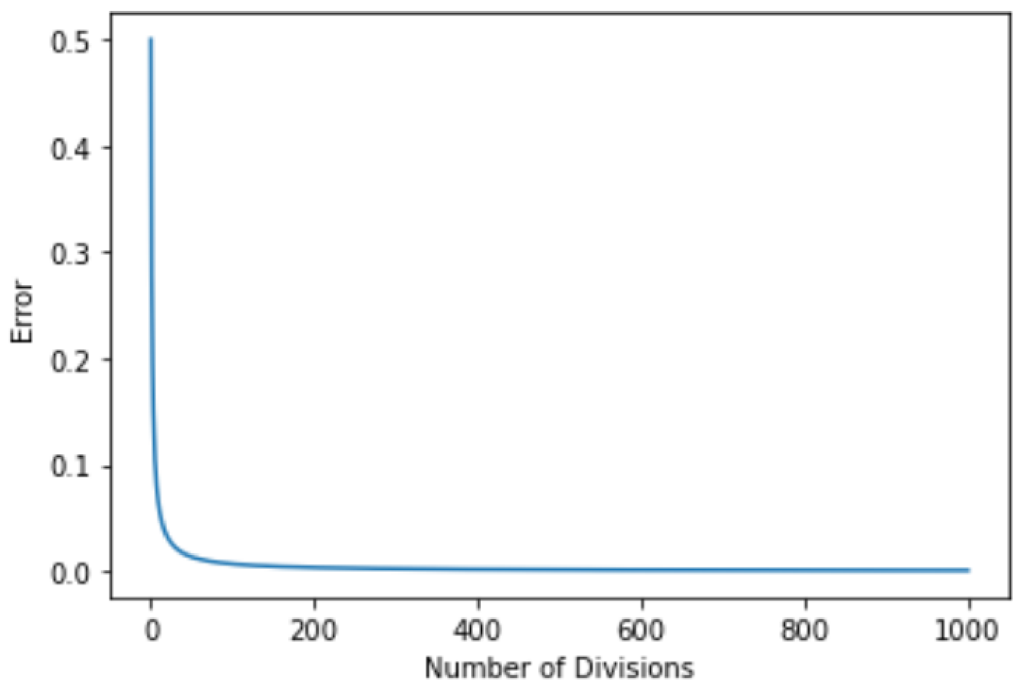
Plot error $e(N)$ versus N.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

arr = []
arr_x = []

for n in range (1,1001):
    arr_x.append(n)
    N = n

    #make the data points
    divisions = np.linspace(0,1,N+1)

    #value of delta
    #it will be distance/N
    Δ = 1/N

    #initilize the array with the zeros
    A = np.zeros((N+1,N+1))

    #put the values of the boundary
    A[0,0] = 1
    A[N,N] = 1

    #due the differential form of boundary condition
    A[N,N-1] = -1

    #make the matrix for the equations
    for i in range(1,N):
        A[i][i] = -2
        A[i][i-1] = 1
        A[i][i+1] = 1


    #solutions of the x
    B = np.zeros(N+1)
    B[1:-1] = -1*Δ*Δ
```

```
    #solve the equation
    y = np.linalg.solve(A,B)

    #make the analytical matrix
    analytical = divisions - ((divisions)**2)/2

    #calculate the numerator error
    error_num = abs(y[1:]-analytical[1:])

    #calcualte the denominator error
    error_den = abs(analytical[1:])

    #calculate the fianl error and add to the array
    error = (np.sum(error_num/error_den))/(N+1)
    arr.append(error)


#plot the array of errors and x
plt.plot(arr_x,arr)
plt.show()
```

Plot Output:

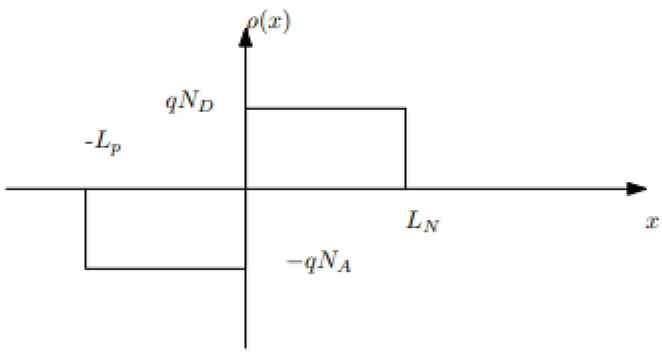


Question 4:(Extra Credits)

4. Consider the following differential equation for a P-N diode with $N_A = N_D = 10^{15}\,\text{cm}^{-3}$, $L_N = L_P = 1\mu m$, and $\epsilon_s = 12\epsilon_0$:

$$\frac{d^2\Phi}{dx^2} = -\frac{\rho(x)}{\epsilon_s}.$$

Solve for $\Phi(x)$ assuming $\Phi(-3L_P) = 0$ and $\Phi'(3L_N) = 0$.



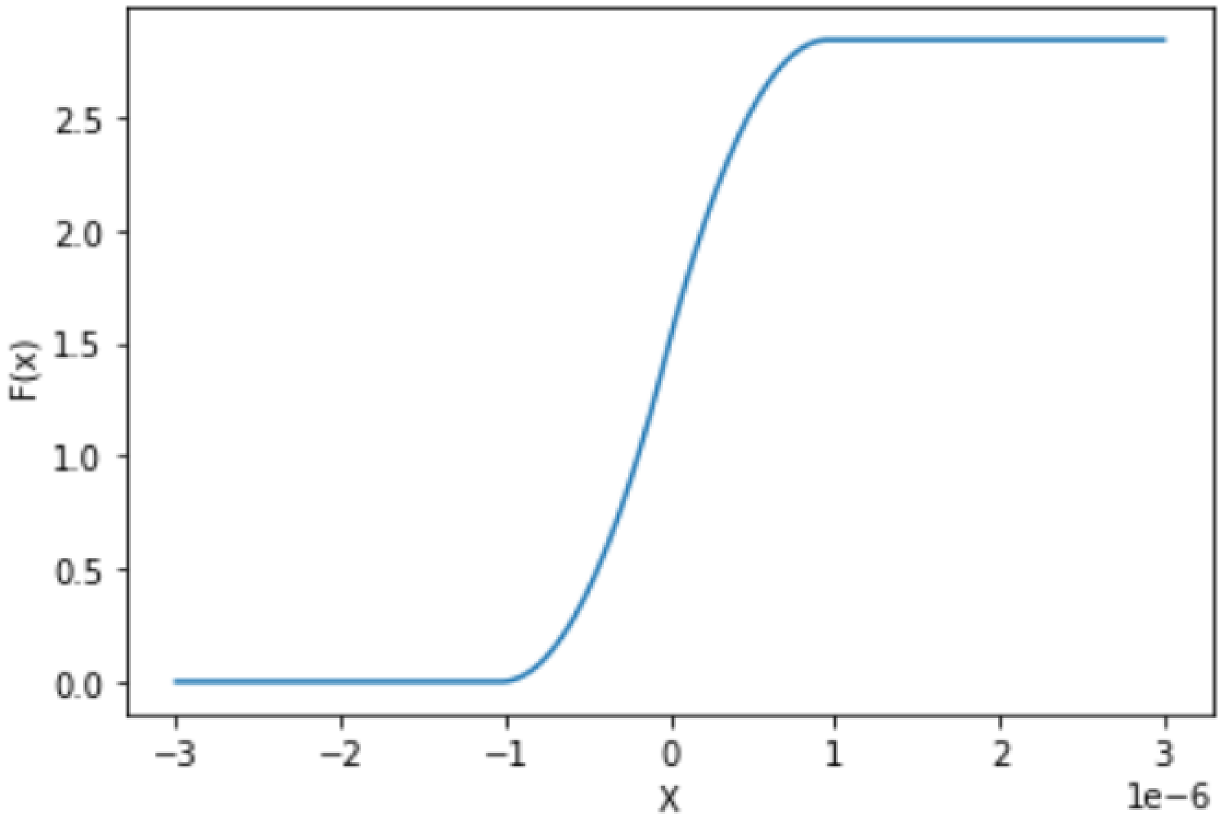
Code:

```
import numpy as np
import matplotlib.pyplot as plt


#definig the constants
Np = (10**(21))
q = 1.6 * ((10)**(-19))
e_s = 12*(8.8**(-12))
Lp = 10**(-6)

#specify the number of points

N = 181

#make the data points

divisions = np.linspace(-3*Lp,3*Lp,N+1)

#value of delta
```

```python
Δ = 6*Lp/N

#initilize the array with the zeros
A = np.zeros((N+1,N+1))

#put the values of the boundary
A[0,0] = 1
A[N,N] = 1

#incorparating the newman boundary condition
A[N,N-1] = -1

#make the matrix for the equations
for i in range(1,N):
    A[i][i] = -2
    A[i][i-1] = 1
    A[i][i+1] = 1

print(A)

#solutions of the x
#divide the function into 3 parts
#<-lp , >lp  , -lp<>lp
p1 = (N+1)//3
p2 = 2*((N+1)//3)

#for first and last part the ans is zero
#for he middle division is further divided into 2 parts
p1_1 = (p1+p2)//2

B = np.zeros(N+1)
B[p1:p1_1] = (Np*Δ*Δ*q/e_s)
B[p1_1:p2] = -(Np*Δ*Δ*q/e_s)

#solve the equation
y = np.linalg.solve(A,B)

#plot the graph
plt.plot(divisions,y)
```

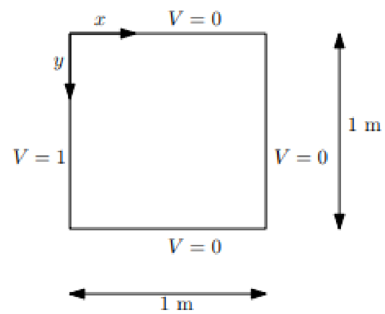
```
plt.xlabel("X")
plt.ylabel("F(x)")
plt.show()
```

Plot Output:



Question 5:

Write a Python program to solve $\nabla^2 V = 0$ with the boundary conditions $V(0, y) = 1$ V, $V(1, y) = 0$ V, $V(x, 0) = 0$ V, and $V(x, 1) = 0$ V.



Compute the error with the analytic solution:

$$V(x,y) = \sum_{n=1,3,5,\ldots} \left(\frac{4}{n\pi \sinh(n\pi)}\right) \sinh(m\pi(1-x)) \sin(m\pi y).$$

Code:

```
import numpy as np
from numpy import sin , sinh , pi
import matplotlib.pyplot as plt

#function to calculate the analytic value at particular point at
particular n
def V(x,y,i):
    ans = ((4/(sinh(i*pi) * i * pi)) * (sinh(i*pi*(1-x))) *
(sin(i*pi*y)))

    return ans

#arrays to store the final error values and points
final_error = []
final_points = []

for N in range(1,100):

    #add the point
    final_points.append(N)
```

```
    #initilize the array with the zeros
    #4 D array will be used
    #each matrix if the 2d array will represent the solution of the
sol(m,n)
    A = np.zeros((N+1,N+1 , N+1,N+1))

    #put the values of the boundary eq
    #initialize the value of the coeficient matrix
    for i in range(N+1):
        A[0][i][0][i] = 1
        A[i][0][i][0] = 1
        A[i][N][i][N] = 1
        A[N][i][N][i] = 1

    #fill the value of the remaining equations
    for i in range(1,N):
        for j in range(1,N):
            A[i,j,i+1,j] = 1
            A[i,j,i,j+1] = 1
            A[i,j,i,j-1] = 1
            A[i,j,i-1,j] = 1
            A[i,j,i,j] = -4

    #the solution matrix will be zero as in question except the
boundary condition
    B = np.zeros((N+1,N+1))
    B[0][1:-1] = 1


    """convert the 4d array into the 2 d array for solving the
equation and convert the
    2d array into the 1 d array"""

    #for A in first row all elements correspondingto the n=0 , next
will be n=1 etx
    A = A.reshape(((N+1)**2,(N+1)**2))

    #for B the first element corresponds to n=0 and so on
```

```
    B = B.reshape(((N+1)**2 ,1))

    #solve the obtained 2d equation
    Sol = np.linalg.solve(A,B)
    Sol = Sol.reshape((N+1,N+1))

    #for the calculation of the error

    #calculate the error for each point in the equation
    points = np.linspace(0,1,N+1)

    #calculate the analytic array
    analytic = np.zeros((N+1,N+1))

    #calculate the analiyitic value for each corresponding point
    for i in range(N+1):
        for j in range(N+1):
            for n in range(1,20,2):

                #call the function to calculate the value
                analytic[i,j] += V(points[i] , points[j] , n)


    #calculate the total error in the graph
    total_error = 0
    for i in range(N+1):
        for j in range(N+1):

                ana = analytic[i,j]
                error_num = abs(Sol[i][j] - ana)
                error_den = abs(ana)

                #to avoid the zero division error get the value to
small number if denominator is zero
                if error_den ==0:
                    error = (error_num/10**(-9))*(1/((N+1)**2))
                else:
                    error = (error_num/error_den)/((N+1)**2)
```

```
                total_error += error

    #add the error into the file
    final_error.append(total_error)

#plotting the error function
plt.plot(final_points,final_error)
plt.xlabel("N")
plt.ylabel("E(N)")
plt.show()
```

Plot Output: