Jacqueline K. Garcia
25070118

April 10, 2017
Nuc E 155, Slaybaugh

Assignment 6

1)

Nuc.E 155 HW 6

* I worked on this problem with Emily Vu

1) $-D \frac{d}{dx} \cdot \frac{d\phi(x)}{dx} + \Sigma_a \phi(x) = S(x)$

B.C. : $\phi(\pm a) = 0$

a) $S(x) = 0$ for $x \in [-a, a]$

$-D \cdot \frac{d}{dx} \frac{d\phi(x)}{dx} = -\Sigma_a \phi(x) \rightarrow D \frac{d^2\phi(x)}{dx^2} - \Sigma_a \phi(x) = 0$

let $\phi(-a) = \phi(a) = 0$

$r^2 - \frac{\Sigma_a}{D} = 0 \rightarrow r = \pm\sqrt{\frac{\Sigma_a}{D}}$

general form: $\phi(x) = C_1 e^{r_1 x} + C_2 e^{r_2 x}$

for $-a$: $0 = C_1 e^{-a\sqrt{\Sigma_a/D}} + C_2 e^{a\sqrt{\Sigma_a/D}}$

for a : $0 = C_1 e^{a\sqrt{\Sigma_a/D}} + C_2 e^{a\sqrt{\Sigma_a/D}}$

$$\rightarrow \begin{bmatrix} e^{-a\sqrt{\Sigma_a/D}} & e^{a\sqrt{\Sigma_a/D}} \\ e^{a\sqrt{\Sigma_a/D}} & e^{-a\sqrt{\Sigma_a/D}} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$C_1 = C_2 = 0$

hence $\phi(x) = 0$

b) $\phi(x) = C_1 e^{x\sqrt{\Sigma_a/D}} + C_2 e^{-x\sqrt{\Sigma_a/D}}$

$\phi(x) = a$ [guess] $\rightarrow \phi'(x) = 0$

$\rightarrow 0 + \Sigma_a (a) = S_0$, $a = \frac{S_0}{\Sigma_a}$

$0 = C_1 e^{a\sqrt{\Sigma_a/D}} + C_2 e^{-a\sqrt{\Sigma_a/D}} + S_0/\Sigma_a \rightarrow -\frac{S_0}{\Sigma_a} = C_1 e^{a\sqrt{\Sigma_a/D}} + C_2 e^{-a\sqrt{\Sigma_a/D}}$

$$\hookrightarrow \begin{bmatrix} e^{\sqrt{\Sigma_a/D}\,a} & e^{-\sqrt{\Sigma_a/D}\,a} \\ e^{-\sqrt{\Sigma_a/D}\,a} & e^{\sqrt{\Sigma_a/D}\,a} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} -S_0/\Sigma_a \\ -S_0/\Sigma_a \end{bmatrix}$$

It appears that $C_1 = C_2$

$C_1 e^{\sqrt{\Sigma_a/D}\,a} + C_1 e^{-\sqrt{\Sigma_a/D}\,a} = -S_0/\Sigma_a$

$\rightarrow C_1 \left(e^{\sqrt{\Sigma_a/D}\,a} + e^{-a\sqrt{\Sigma_a/D}}\right) = -S_0/\Sigma_a$

$$C_1 = \frac{-S_0}{\Sigma_a\left(e^{\sqrt{\Sigma_a/D}\,a} + e^{-\sqrt{\Sigma_a/D}\,a}\right)} \cdot \frac{e^{\sqrt{\Sigma_a/D}\,a}}{e^{\sqrt{\Sigma_a/D}\,a}}$$

$$\rightarrow C_1 = \frac{-S_0 e^{\sqrt{\Sigma_a/D}\,a}}{\Sigma_a\left(e^{2\sqrt{\Sigma_a/D}\,a} + 1\right)}$$

$$\phi(x) = \frac{-S_0 e^{a\sqrt{\Sigma_a/D}}}{\Sigma_a\left(e^{2a\sqrt{\Sigma_a/D}} + 1\right)} \left[e^{x\sqrt{\Sigma_a/D}} + e^{-x\sqrt{\Sigma_a/D}}\right] + \frac{S_0}{\Sigma_a}$$

c) $S(x) = \cos(x)$ for $x \in [-a, a]$

$\phi(x) = a\cos(x) + b\sin(x)$

$\Rightarrow \frac{d\phi(x)}{dx} = -a\sin(x) + b\cos(x) \Rightarrow \frac{d^2\phi(x)}{d^2x} = -a\cos(x) - b\sin(x)$

$\Rightarrow D \cdot a\cos(x) + D \cdot b\sin(x) + \Sigma_a a\cos(x) + \Sigma_a b\sin(x) = \cos(x)$

* since there are no sin(x)'s, $b = 0$

$\cancel{\cos(x)}[Da + \Sigma_a a] = \cancel{\cos(x)}$

$\Rightarrow D \cdot a + \Sigma_a \cdot a = 1$

$\Rightarrow a(D + \Sigma_a) = 1; \quad a = \frac{1}{D + \Sigma_a}$

$\phi(x) = \left(\frac{1}{D + \Sigma_a}\right)\cos(x)$

$$0 = C_1 e^{\sqrt{\Sigma_a/D}\,a} + C_2 e^{-\sqrt{\Sigma_a/D}\,a} + \left(\frac{1}{D + \Sigma_a}\right)\cos(a)$$

$$C_1 e^{\sqrt{\Sigma_a/D}\,a} + C_2 e^{-\sqrt{\Sigma_a/D}\,a} = -\frac{1}{(D + \Sigma_a)}\cos(a)$$

$$\begin{bmatrix} e^{\sqrt{\Sigma_a/D}\,a} & e^{-\sqrt{\Sigma_a/D}\,a} \\ e^{-\sqrt{\Sigma_a/D}\,a} & e^{\sqrt{\Sigma_a/D}\,a} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{(D+\Sigma_a)}\cos(a) \\ -\frac{1}{(D+\Sigma_a)}\cos(a) \end{bmatrix} \qquad C_1 = C_2$$

$$C_1\left(e^{\sqrt{\Sigma_a/D}\,a} + e^{-\sqrt{\Sigma_a/D}\,a}\right) = -\frac{1}{(D + \Sigma_a)}\cos(a)$$

$$C_1 = -\frac{1}{(D + \Sigma_a)} \cdot \frac{\cos(a) \cdot e^{\sqrt{\Sigma_a/D}\,a}}{\left(e^{2\sqrt{\Sigma_a/D} \cdot a} + 1\right)}$$

$$\phi(x) = -\frac{1}{(D + \Sigma_a)}\left[\frac{\cos(a)e^{\sqrt{\Sigma_a/D} \cdot a}}{\left(e^{2\sqrt{\Sigma_a/D}\,a} + 1\right)}\right]\left(e^{\sqrt{\Sigma_a/D}\,x} + e^{-\sqrt{\Sigma_a/D}\,x}\right) + \left(\frac{1}{D + \Sigma_a}\right)\cos(x)$$

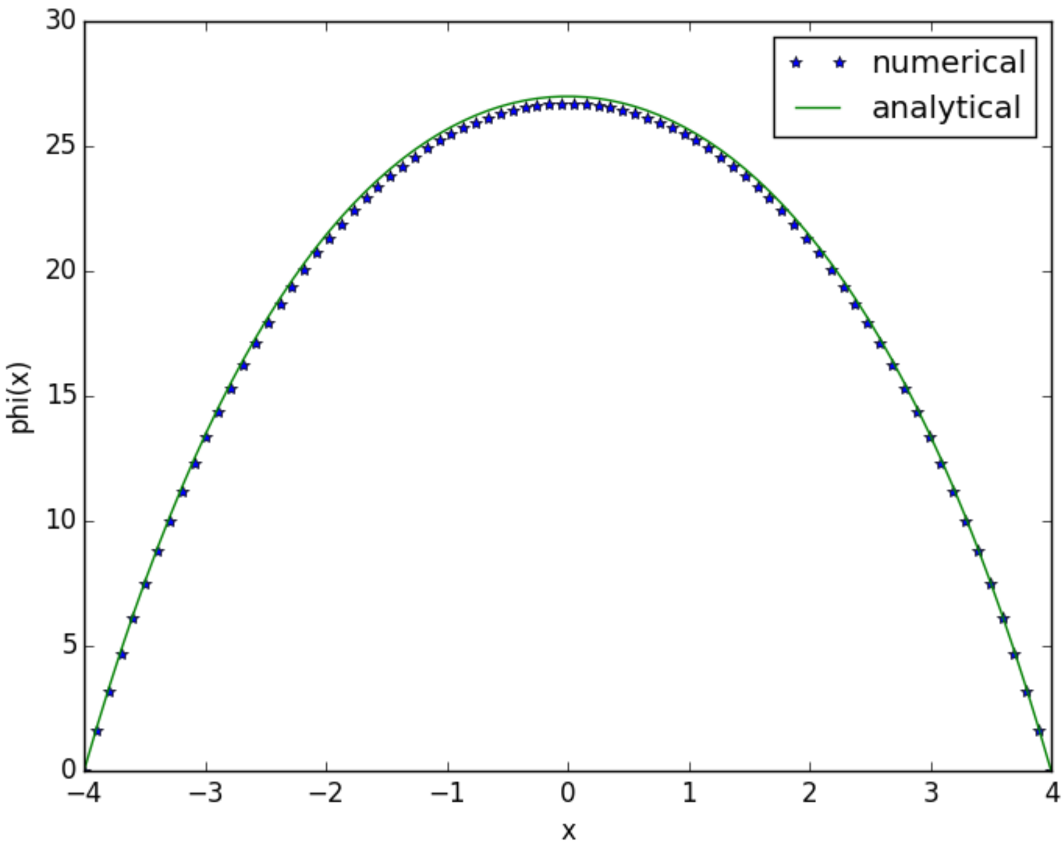
2)

Code: *NE155_hw6_2.py*

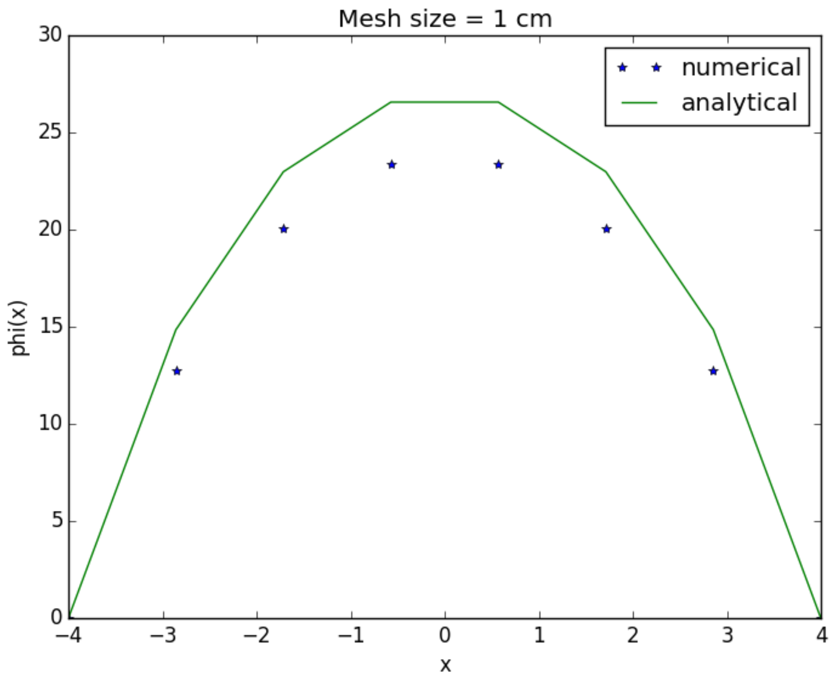
```
import numpy as np
import matplotlib.pyplot as plt

a = 4 #values are in cm
D = 1
Sigma = 0.2 # 1/cm
S = 8 # n/(cm^3*s)
h = 0.1 #cm
L = (D/Sigma)**0.5
```

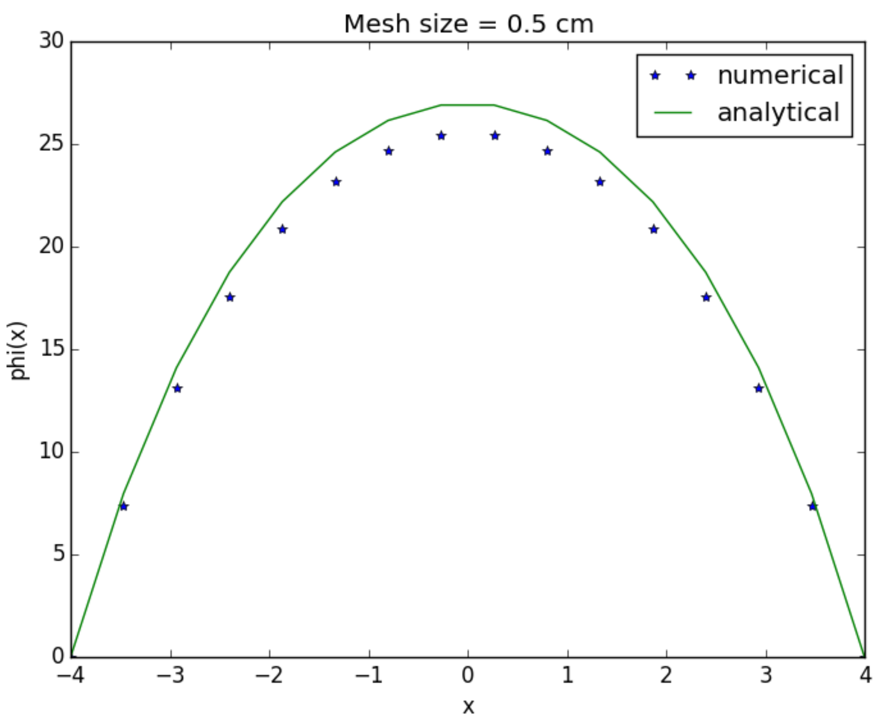
```
b_array = np.zeros((78, 1))
i = 0
while i < 78:
    b_array[i][0] = 8
    i += 1

A = -D/(h ** 2)
B = D * (2 + (h**2)/(L**2))/(h**2)
C = -D/(h**2)

def matrix_a():
    a = []
    c = []
    i = 0
    while i < 77:
        a.append(A)
        c.append(C)
        i += 1
    b = []
    i = 0
    while i < 78:
        b.append(B)
        i += 1

    def tridiag(a, b, c, k1=-1, k2=0, k3=1):
        return np.diag(a, k1) + np.diag(b, k2) + np.diag(c, k3)
    M = tridiag(a, b, c)
    return M

Am = matrix_a()
print(Am)

sol = np.dot(np.linalg.inv(Am), b_array)
print(sol)

new_sol = []
new_sol.append(0)
for i in sol:
    new_sol.append(i)
new_sol.append(0)

x_axis = np.linspace(-4, 4, 80)
phi_val = []
for i in x_axis:
    phi = (-8*np.exp(a*(Sigma/D)**0.5))/(Sigma*(np.exp(2*a*(Sigma/D)**0.5)+1))*(np.exp(i*(Sigma/D)**0.5)+np.exp(-i*(Sigma/D)**0.5)) + 8/Sigma
    phi_val.append(phi)

plt.plot(x_axis, new_sol, '*')
plt.plot(x_axis, phi_val)
```

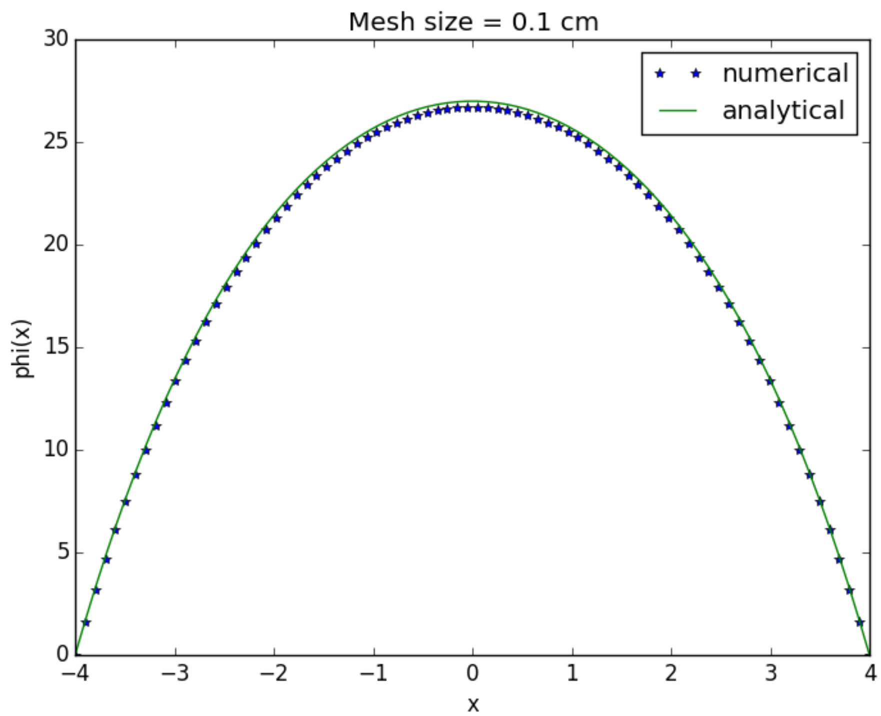
```
plt.xlabel('x')
plt.ylabel('phi(x)')
plt.legend(('numerical', 'analytical'))
plt.show()
```

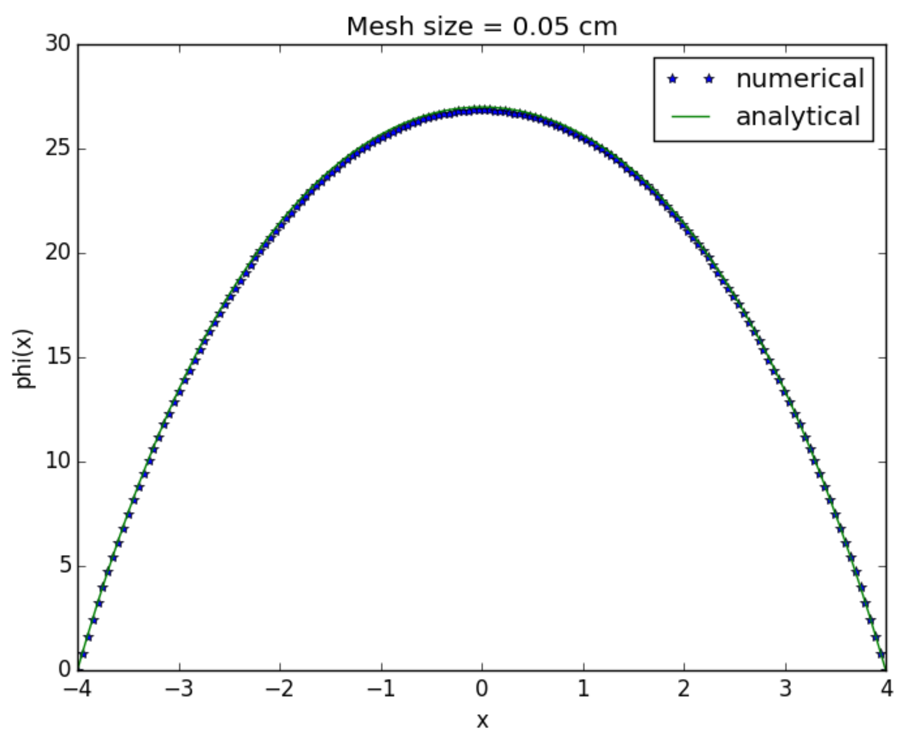


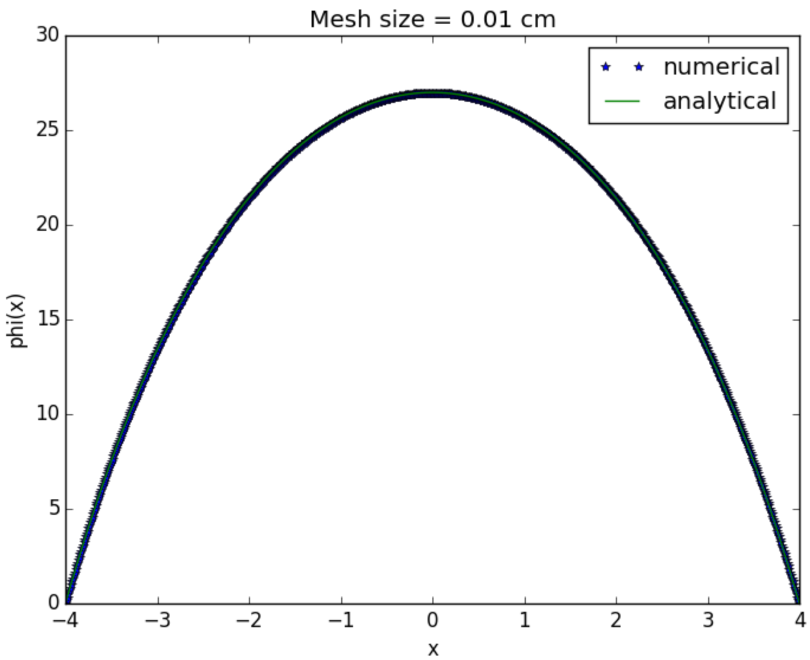
3)

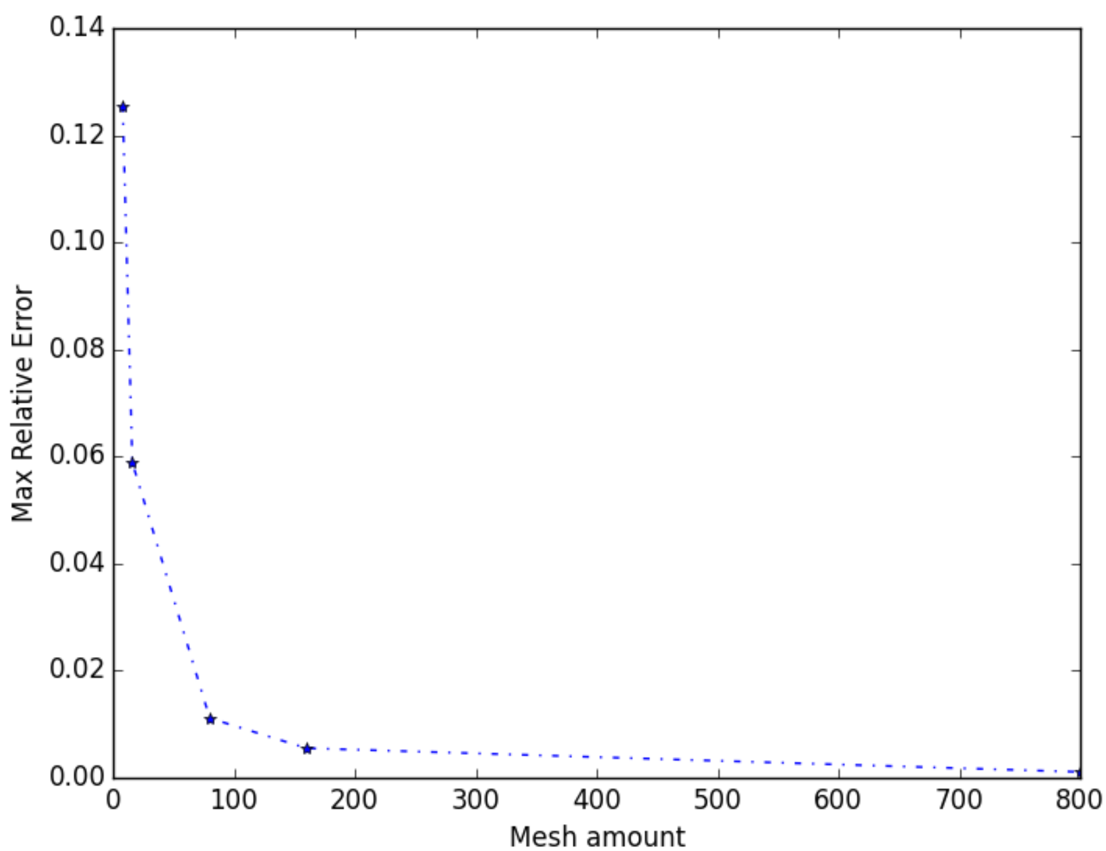

Mesh size = 1 cm
numerical
analytical
phi(x)
x


Mesh size = 0.5 cm
numerical
analytical
phi(x)
x


Mesh size = 0.1 cm
numerical
analytical
phi(x)
x
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4


Mesh size = 0.05 cm
numerical
analytical
phi(x)
x
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4


Mesh size = 0.01 cm
numerical
analytical
phi(x)
x
30
25
20
15
10
5
0
−4
−3
−2
−1
0
1
2
3
4


Max Relative Error
Mesh amount
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
100
200
300
400
500
600
700
800

The higher the mess amount, the less the max relative error becomes. The reason for this is because decreasing the mesh size increases the number of calculation which in turn increases the accuracy. We can also see that there is an exponential trend between the mesh amount and the relative error. This is because the mesh size is present as h^2 in our equation.

4)

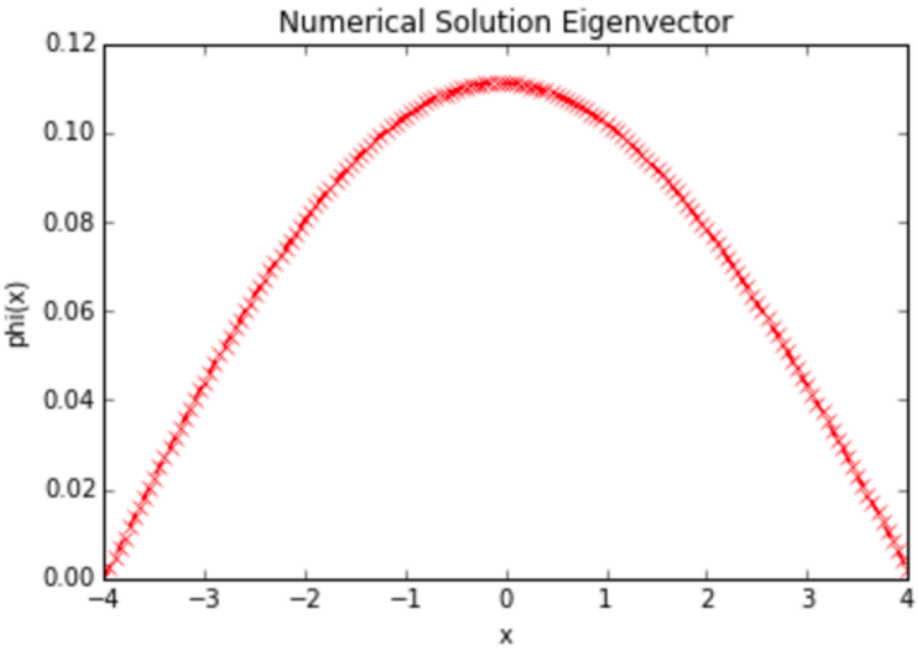
Code: NE155_hw6_4

```
import matplotlib.pyplot as plt
import numpy as np
import Ne155_hw5_4 as It_sol
import copy

SigA =0.7
vSig = 0.6
h = 0.1
D = 1
a = 4
S = 8

def makeQf = (vSig, phi):
    Qf = vSig * phi
    return Qf

def makeA(h, n, D, SigA):
    row_1 = -D / (h ** 2)
    row_2 = (2 * D)/( h** 2) + SigA
    row_3 = -D/(h ** 2)
    values = [row_1, row_2, row_3]
    diag = [-1, 0, 1]
    A = sci.sparse.diags(values, diag, shape = (n+1, n+1)).todense()
    #Adjust A for boundary conditions
    A[0, 0] = 1
    A[0, 1] = 0
    A[n, n] = 1
    A[n, n-1] = 0
    return A

#phi_in is the initial guess
def IterSOl(SigA, vSig, h, D, phi_in, tol, k_0, k_tol, absolute = False):
    n = np.size(phi_in)

    phi_n = np.zeros(n)
    phi_old = phi_in / np.linalg.norm(phi_in)
    k_new = 0
    k_old = k_0

    A = makeA(h, n, D, SigA)
    Qf_old = makeQf(vSig, phi_in)

    phi_error = 0
    k_error = 1
    max_it = 100
    eigen_it = 0
    GS_it = [0]
```

```
    while ((phi_error > tol) and (eigen_it < max_it) and (k_error > k_tol)):
        eigen_it = eigen_it + 1

        (phi_n, GS_error, phi_it) = It_sol.Gauss_Seidel()
        #GS has been modified in hw 5 code to take in the inputs for this
problem
        GS_it.append(phi_it)

        Qf_new = makeQf(vSig, phi_n)
        k_new = k_old * (np.sum(Qf_new) / np.sum(Qf_old))

        if (absolute == True):
            phi_error = np.linalg.norm(np.abs(phi_n - phi_old), 2)
            k_error = np.abs(k_new - k_old)
        else:
            phi_error = np.linalg.norm(np.abs(phi_n - phi_old), 2) /
np.linalg.norm(phi_new, 2)
            k_error = np.abs(k_new - k_old)

        phi_old = copy.deecopy(phi_new)
        k_old = copy.deecopy(k_new)
        Qf_old = copy.deecopy(Qf_new)

    phi_n = phi_n / np.linalg.norm(phi_n)
    return("k": k_new, "k_error": k_error, "phi": phi_n, "phi_error":
phi_error, "GS Iterations": GS_it, "Power Iterations": eigen_it)

#Initialize guesses:
phi_in = np.ones(n + 1)
k_0 = 1
tol = 1 * 10 **(-4)
k_tol = 1 * 10 **(-4)

sol = IterSol(SigA, vSig, h, D, phi_in, tol, k_0, k_tol, True)
print(sol["k"], sol["Power Iterations"])

x_axis = np.linspace(-a, a, n + 1)
plt.plot(x_axis, sol["phi"], 'r--x')
plt.xlabel('x')
plt.ylabel('phi(x)')
plt.title('Numerical Solution Eigenvector')
plt.show
```

Output:

```
0.812465570786
20
```


Numerical Solution Eigenvector
phi(x)
x
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−4
−3
−2
−1
0
1
2
3
4