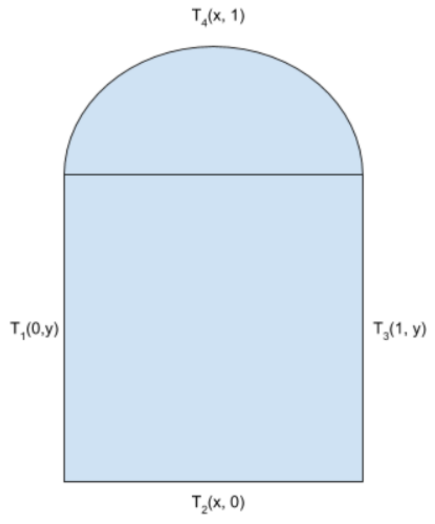
# <u>AERO 430 – Exam 2</u>

Antonio Diaz '22

UIN 327003625

**Due 04/20/2020**

## 1) Analytical Solution to 2D Heat Equation for infinitely long plate.



Case 1 Boundary conditions:

$$T_1(0,y) = T_2(x,0) = T_3(1,y) = 0° \, C$$
$$T_4(x,1) = 100 \ \cdot \sin(\pi x)° \, C$$

Case 2 Boundary conditions:

$$T_1(0,y) = T_2(x,0) = T_3(x,1) = 0° \, C$$
$$T_4(x,1) = \frac{du}{dy} = K\pi \cosh(K\pi) \frac{100 \sin(\pi x)}{\sinh(K\pi)}$$ , where K is a conductivity constant.

Constants are defined as follows:

$$K = .5 \frac{cal}{sec \cdot C \cdot cm)}$$

Length = 1 cm

Radius = .1 cm

The governing equation describing heat flow across any surface is $\nabla Q = 0$.
Following the conservation of heat flow, Q is defined as:

$$Q = -K_x \frac{du}{dx} \, \hat{\imath} - K_y \frac{du}{dy} \, \hat{k}$$

$$\nabla Q = -K_{xx} \frac{d^2u}{d^2x} \, \hat{\imath} - K_{yy} \frac{d^2u}{d^2y} \, \hat{k} = 0$$

This can be reduced by simplifying the K constants, $\frac{K_{xx}}{K_{yy}} = K^2$

$$K^2 \frac{d^2u}{d^2x} (x,y) + \frac{d^2u}{d^2y} (x,y) = 0$$

**Separation of variables to solve Dirichlet 2nd order homogenous differential equation:**

The heat function $u(x, y)$ can be separated into $f(x)g(y)$. Plugging back into PDE:

$$\frac{1}{f}\cdot\frac{d^2f}{dx^2}+\frac{1}{K^2}\frac{1}{g}\cdot\frac{d^2g}{dy^2}=0 \text{ or } -\frac{1}{f}\cdot\frac{d^2f}{dx^2}=\frac{1}{K^2}\frac{1}{g}\cdot\frac{d^2g}{dy^2}=\lambda$$

Boundary Conditions for Case 1:

$f(0)g(y) = f(1)g(y) = 0°\,C$

$f(x)g(0) = 0°\,C, f(x)g(1) = 100\ \cdot\sin(\pi x)°\,C$

Boundary Conditions for Case 2:

$f(0)g(y) = f(1)g(y) = 0°\,C$

$f(x)g(0) = 0°\,C, f(x)g'(1) = K\pi\cosh(K\pi)\frac{100\sin(\pi x)}{\sinh(K\pi)}°\,C$

General solution to $\frac{d^2f}{dx^2} = -f\lambda$:

$f(x) = A cos(kx) + B sin(kx)$

$f(0) = f(1) = 0$ from boundary conditions

$f(0) = A cos(0) + B sin(0) = A cos(0) = 0$ gives us A = 0

$f(1) = 0 cos(k) + B sin(k) = B sin(k) = 0$ gives us $k = n\pi$, n is any integer

This gives us $f(x) = sin(\pi x)$

General solution to $\frac{d^2g}{dy^2} = K^2 g\lambda$:

$g(y) = Ae^{n\pi y} + Be^{-n\pi y}$

$g(0) = 0, f(x)g(1) = 100\cdot\sin(\pi x)\ °C$ from boundary conditions

$g(0) = A + B = 0$

$g(1) = Ae^{\pi} + Be^{-\pi} = 100$

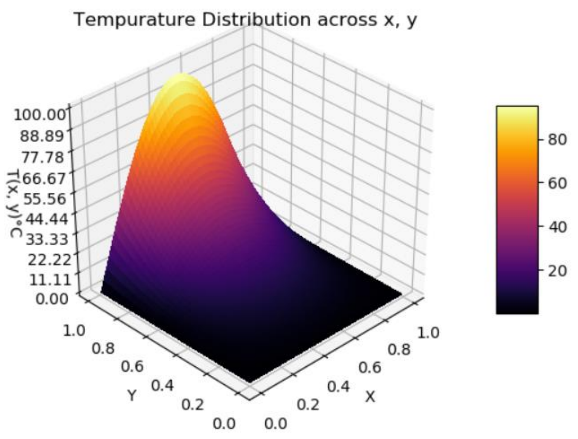
This system of equations combined with boundary conditions reduces to:

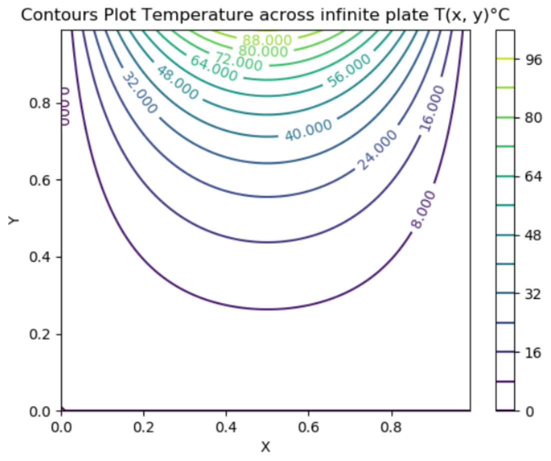
$g(y) = \frac{100\sinh(K\pi y)}{\sinh(K\pi)}$

Final General Solution to Heat Equation:

$$u(x, y) = f(x)g(y) = \boldsymbol{\frac{100\sinh(K\pi y)}{\sinh(K\pi)}\cdot sin(\pi x)}$$

This heat temperature function is graphed below:





## 2) FDM for point $x_i, y_i$:

Ordinary Taylor Series Expansion:

$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{3!} f'''(x) + \frac{h^4}{4!} f''''(x)$ + Error

$f(x+0) = f(x)$

$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{3!} f'''(x) + \frac{h^4}{4!} f''''(x)$ + Error

Taylor series expansion for point:

$U_{i-\Delta x,j} = U_{i,j} - \Delta x U'_{i,j} + \frac{\Delta x^2}{2} U''_{i,j} - \frac{\Delta x^3}{2} U'''_{i,j} + \frac{\Delta x^4}{2} U''''_{i,j} + O(\Delta x^4)$

$U_{i+0,j} = U_{i,j}$

$U_{i+\Delta x,j} = U_{i,j} + \Delta x U'_{i,j} + \frac{\Delta x^2}{2} U''_{i,j} + \frac{\Delta x^3}{2} U'''_{i,j} + \frac{\Delta x^4}{2} U''''_{i,j} + O(\Delta x^4)$

$U_{i,j-\Delta y} = U_{i,j} - \Delta y U'_{i,j} + \frac{\Delta y^2}{2} U''_{i,j} - \frac{\Delta y^3}{2} U'''_{i,j} + \frac{\Delta y^4}{2} U''''_{i,j} + O(\Delta y^4)$

$U_{i,j+0} = U_{i,j}$

$U_{i,j+\Delta y} = U_{i,j} + \Delta y U'_{i,j} + \frac{\Delta y^2}{2} U''_{i,j} + \frac{\Delta y^3}{2} U'''_{i,j} + \frac{\Delta y^4}{2} U''''_{i,j} + O(\Delta y^4)$

Subtracting Taylor series to find the second order approximation of U':

$$U_{i,j+\Delta y} - U_{i,j-\Delta y} =$$

$U_{i,j} - U_{i,j} + \Delta y U'_{i,j} + \Delta y U'_{i,j} + \frac{\Delta y^2}{2} U''_{i,j} - \frac{\Delta y^2}{2} U''_{i,j} + \frac{\Delta y^3}{2} U'''_{i,j} + \frac{\Delta y^3}{2} U'''_{i,j} + \frac{\Delta y^4}{2} U''''_{i,j} - \frac{\Delta y^4}{2} U''''_{i,j} = 2\Delta y U'_{i,j} + \frac{\Delta y^3}{1} U'''_{i,j} + O(\Delta y^4)$

Dividing both sides by $2\Delta y$ gives us $\frac{dU}{dy}, \frac{U_{i,j+\Delta y} - U_{i,j-\Delta y}}{2\Delta y} \approx U'_{i,j}$

Similarly, for $\frac{dU}{dx}, \frac{U_{i,j+\Delta x} - U_{i,j-\Delta x}}{2\Delta x} \approx U'_{i,j}$

Adding Taylor series to find the second order approximation of U'':

$$U_{i,j+\Delta y} + U_{i,j-\Delta y} = 2U_{i,j} + \frac{\Delta y^2}{1} U''_{i,j} + \frac{\Delta y^4}{12} U''''_{i,j} + O(\Delta y^4)$$

$$U_{i,j+\Delta x} + U_{i,j-\Delta x} = 2U_{i,j} + \frac{\Delta x^2}{1} U''_{i,j} + \frac{\Delta x^4}{12} U''''_{i,j} + O(\Delta x^4)$$

This is then reduced after removing truncation error, becomes:

$$\frac{U_{i,j+\Delta y} + U_{i,j-\Delta y} - 2U_{i,j}}{\Delta y^2} \approx \frac{d^2U}{dy^2}$$

$$\frac{U_{i,j+\Delta x} + U_{i,j-\Delta x} - 2U_{i,j}}{\Delta x^2} \approx \frac{d^2U}{dx^2}$$

Using the Taylor series similar for the fourth order approximation of U'':

$$\frac{U_{i,j+\Delta y} + U_{i,j-\Delta y} - 2U_{i,j}}{\Delta y^2} - \frac{\Delta y^2}{12} \frac{d^4U}{dy^4} \approx \frac{d^2U}{dy^2}$$

$$\frac{U_{i,j+\Delta x} + U_{i,j-\Delta x} - 2U_{i,j}}{\Delta x^2} - \frac{\Delta x^2}{12} \frac{d^4U}{dx^4} \approx \frac{d^2U}{dx^2}$$

Simplifying these expressions using the heat equations:

$$K^2 \frac{\partial^2 u}{\partial^2 x}\ (x,y) + \frac{\partial^2 u}{\partial^2 y}\ (x,y) = 0$$

$$K^2 \frac{\partial^4 u}{\partial^2 y \partial^2 x}\ (x,y) = -\frac{\partial^4 u}{\partial y^4}\ (x,y)$$

$$K^2 \frac{\partial^4 u}{\partial^4 x}\ (x,y) = -\frac{\partial^4 u}{\partial x^2 \partial y^2}\ (x,y)$$

$$\frac{U_{i,j+\Delta y} + U_{i,j-\Delta y} - 2U_{i,j}}{\Delta y^2} + K^2 \frac{\Delta y^2}{12} \frac{\partial^4 u}{\partial^2 y \partial^2 x}\ (x,y) \approx \frac{\partial^2 U}{\partial y^2}$$

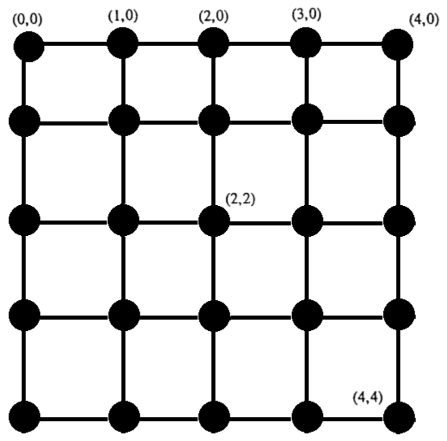
$$\frac{U_{i,j+\Delta x} + U_{i,j-\Delta x} - 2U_{i,j}}{\Delta x^2} + \frac{1}{K^2} \frac{\Delta x^2}{12} \frac{\partial^4 u}{\partial x^2 \partial y^2} \approx \frac{\partial^2 U}{\partial x^2}$$

$$\frac{\partial^4 y}{\partial y^2 \partial x^2} =$$

$$= \frac{(U_{i-\Delta x,j-\Delta y} + U_{i+\Delta x,j-\Delta y} - 2U_{i,j-\Delta y}) + 2(U_{i-\Delta x,j+\Delta y} + U_{i+\Delta x,j} - 2U_{i,j}) + (U_{i-\Delta x,j+\Delta y} + U_{i+\Delta x,j+\Delta y} - 2U_{i,j+\Delta y})}{\Delta \mathrm{x}^2\ \Delta y^2}$$

## 3. Application of FDM across 2D mesh:

A uniform 2D mesh of $\Delta x\ and\ \Delta y = .25$ is used to create the following:



Applying the conditions for approximating the heat equation:

$$K^2 \frac{\partial^2 u}{\partial^2 x}\ (x,y) + \frac{\partial^2 u}{\partial^2 y}\ (x,y) = 0$$

Using previously derived second order approximation

$$K^2 \frac{U_{i,j+\Delta x} + U_{i,j-\Delta x} - 2U_{i,j}}{\Delta x^2} + \frac{U_{i,j+\Delta y} + U_{i,j-\Delta y} - 2U_{i,j}}{\Delta y^2} = 0$$

Using $\frac{\Delta x}{\Delta y} = 1$, this can be reduced to:

$$-2(K^2+1)U_{i,j} + K^2(U_{i,j+\Delta x} + U_{i,j-\Delta x}) + U_{i,j+\Delta y} + U_{i,j-\Delta y} = 0$$

Applying system of equations to $\Delta x = \Delta y = .5$:

$$-2(K^2+1)U_{\Delta x,\Delta y} + K^2(0+0) + 100 + 0 = 0$$
$$U_{\Delta x,\Delta y} = \frac{100}{2(K^2+1)}$$

Applying system of equations to $\Delta x = \Delta y = .25$: for shown mesh:

$$-2(K^2+1)T_{1,1} + K^2(T_{2,1} + T_{0,1}) + \sin(\pi * .25) + T_{1,2} = 0$$
$$-2(K^2+1)T_{2,1} + K^2(T_{3,1} + T_{1,1}) + \sin(\pi * .5) + T_{2,2} = 0$$
$$-2(K^2+1)T_{3,1} + K^2(T_{4,1} + T_{2,1}) + \sin(\pi * .25) + T_{3,2} = 0$$

This is then reduced to a global matrix that solved for each temperature node:

$$\begin{bmatrix} -2(K^2+1) & K^2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ K^2 & -2(K^2+1) & K^2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & K^2 & -2(K^2+1) & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -2(K^2+1) & K^2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & K^2 & -2(K^2+1) & K^2 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & K^2 & -2(K^2+1) & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -2(K^2+1) & K^2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & K^2 & -2(K^2+1) & K^2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & K^2 & -2(K^2+1) \end{bmatrix} \begin{bmatrix} T_{11} \\ T_{21} \\ T_{31} \\ T_{12} \\ T_{22} \\ T_{32} \\ T_{13} \\ T_{23} \\ T_{33} \end{bmatrix} = \begin{bmatrix} -70.71067812 \\ -100 \\ -70.71067812 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

*note that order of elements is from top to right and down in mesh. Some other published results may start from bottom right of nodes in mesh.

4th order FDM is similarly developed as follows:

Applying the conditions for approximating the heat equation:

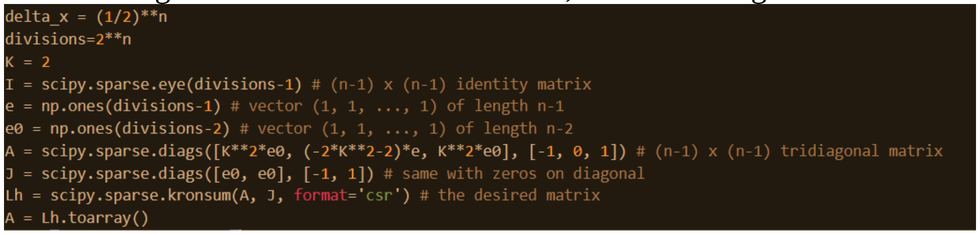
$$K^2 \frac{\partial^2 u}{\partial^2 x}(x,y) + \frac{\partial^2 u}{\partial^2 y}(x,y) = 0$$

Using previously derived 4th order approximation:

$$K^2\left(\frac{U_{i,j+\Delta x} + U_{i,j-\Delta x} - 2U_{i,j}}{\Delta x^2} + \frac{1}{K^2}\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^2 \partial y^2}\right) + \left(\frac{U_{i,j+\Delta y} + U_{i,j-\Delta y} - 2U_{i,j}}{\Delta y^2} + K^2 \frac{\Delta y^2}{12}\frac{\partial^4 u}{\partial^2 y \partial^2 x}\right) = 0$$

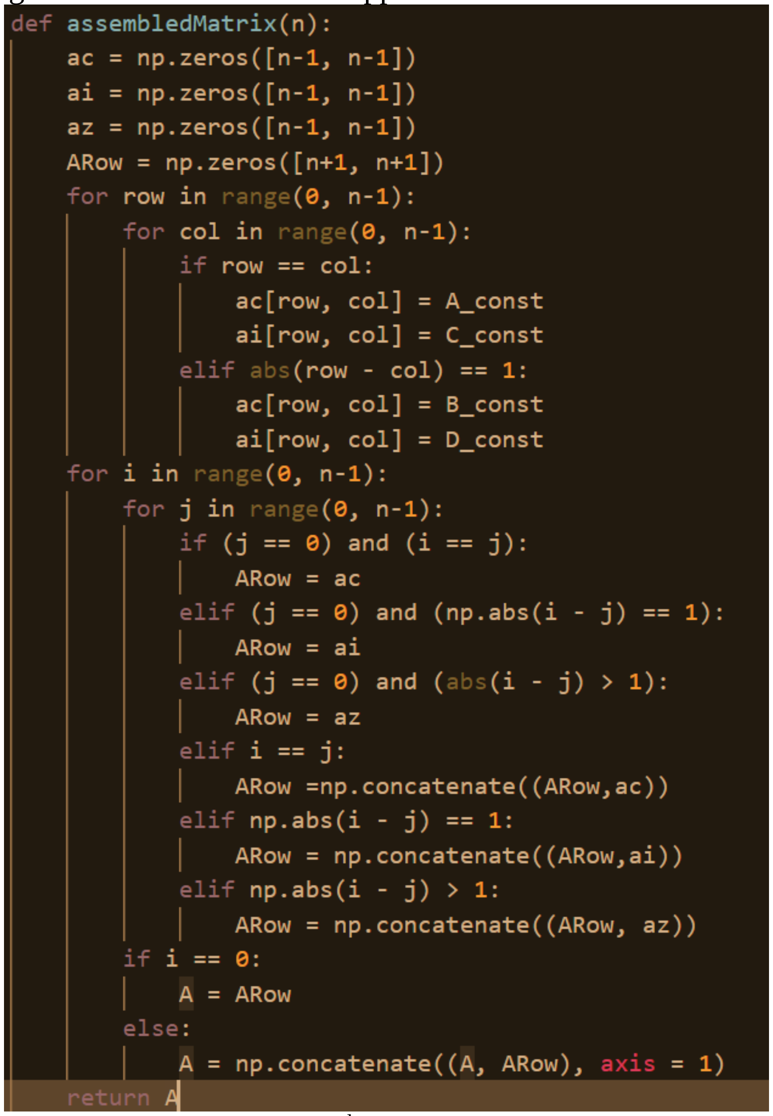
Summary of python code for 2nd order approximation:

Code to create global matrix A as function of n, where n is degree from $.5^n$:



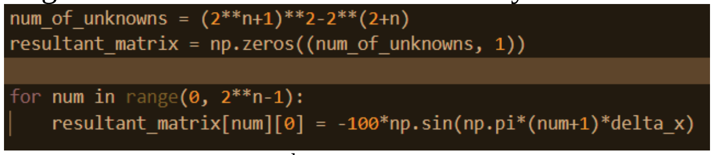
```
delta_x = (1/2)**n
divisions=2**n
K = 2
I = scipy.sparse.eye(divisions-1) # (n-1) x (n-1) identity matrix
e = np.ones(divisions-1) # vector (1, 1, ..., 1) of length n-1
e0 = np.ones(divisions-2) # vector (1, 1, ..., 1) of length n-2
A = scipy.sparse.diags([K**2*e0, (-2*K**2-2)*e, K**2*e0], [-1, 0, 1]) # (n-1) x (n-1) tridiagonal matrix
J = scipy.sparse.diags([e0, e0], [-1, 1]) # same with zeros on diagonal
Lh = scipy.sparse.kronsum(A, J, format='csr') # the desired matrix
A = Lh.toarray()
```

Assembled global matrix for 4th order approximation:



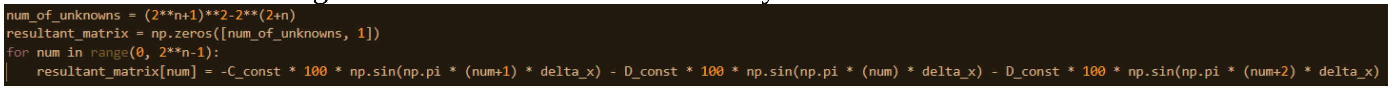
```
def assembledMatrix(n):
    ac = np.zeros([n-1, n-1])
    ai = np.zeros([n-1, n-1])
    az = np.zeros([n-1, n-1])
    ARow = np.zeros([n+1, n+1])
    for row in range(0, n-1):
        for col in range(0, n-1):
            if row == col:
                ac[row, col] = A_const
                ai[row, col] = C_const
            elif abs(row - col) == 1:
                ac[row, col] = B_const
                ai[row, col] = D_const
    for i in range(0, n-1):
        for j in range(0, n-1):
            if (j == 0) and (i == j):
                ARow = ac
            elif (j == 0) and (np.abs(i - j) == 1):
                ARow = ai
            elif (j == 0) and (abs(i - j) > 1):
                ARow = az
            elif i == j:
                ARow =np.concatenate((ARow,ac))
            elif np.abs(i - j) == 1:
                ARow = np.concatenate((ARow,ai))
            elif np.abs(i - j) > 1:
                ARow = np.concatenate((ARow, az))
        if i == 0:
            A = ARow
        else:
            A = np.concatenate((A, ARow), axis = 1)
    return A
```

Code to create right hand side with 2nd order boundary conditions:



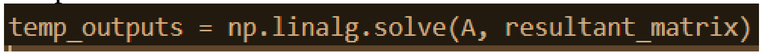
```
num_of_unknowns = (2**n+1)**2-2**(2+n)
resultant_matrix = np.zeros((num_of_unknowns, 1))

for num in range(0, 2**n-1):
    resultant_matrix[num][0] = -100*np.sin(np.pi*(num+1)*delta_x)
```

Code to create right had side with 4th order boundary conditions:



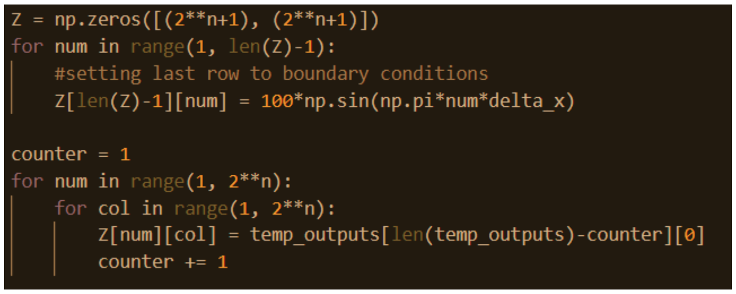
```
num_of_unknowns = (2**n+1)**2-2**(2+n)
resultant_matrix = np.zeros([num_of_unknowns, 1])
for num in range(0, 2**n-1):
    resultant_matrix[num] = -C_const * 100 * np.sin(np.pi * (num+1) * delta_x) - D_const * 100 * np.sin(np.pi * (num) * delta_x) - D_const * 100 * np.sin(np.pi * (num+2) * delta_x)
```

Solving for temperature matrix:



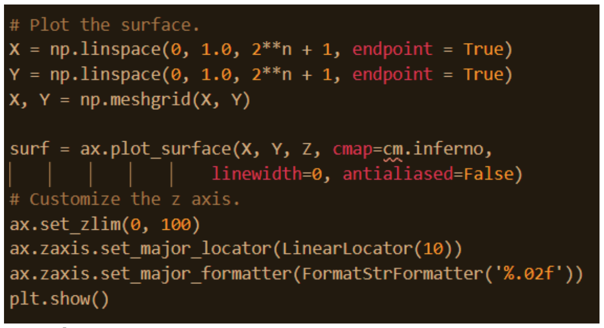
```
temp_outputs = np.linalg.solve(A, resultant_matrix)
```

Code for mapping temperature matrix to mesh for matplotlib to plot:



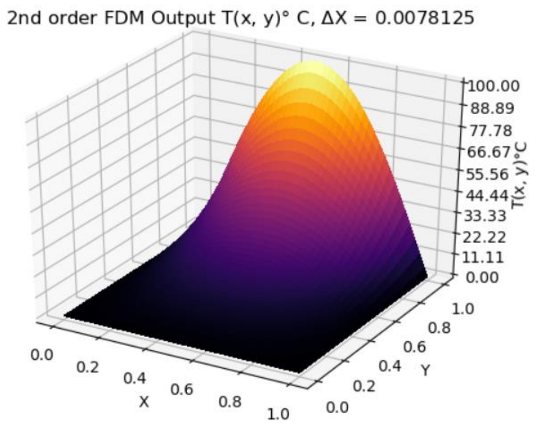
```
Z = np.zeros([(2**n+1), (2**n+1)])
for num in range(1, len(Z)-1):
    #setting last row to boundary conditions
    Z[len(Z)-1][num] = 100*np.sin(np.pi*num*delta_x)

counter = 1
for num in range(1, 2**n):
    for col in range(1, 2**n):
        Z[num][col] = temp_outputs[len(temp_outputs)-counter][0]
        counter += 1
```

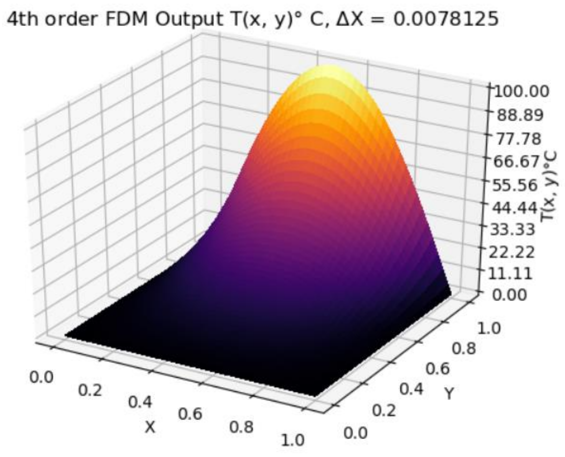
Plotting x, y, z, mesh from finite difference:



```
# Plot the surface.
X = np.linspace(0, 1.0, 2**n + 1, endpoint = True)
Y = np.linspace(0, 1.0, 2**n + 1, endpoint = True)
X, Y = np.meshgrid(X, Y)

surf = ax.plot_surface(X, Y, Z, cmap=cm.inferno,
                       linewidth=0, antialiased=False)
# Customize the z axis.
ax.set_zlim(0, 100)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
plt.show()
```

2nd and 4th order FDM output for 2**7 divisions, K = .7:

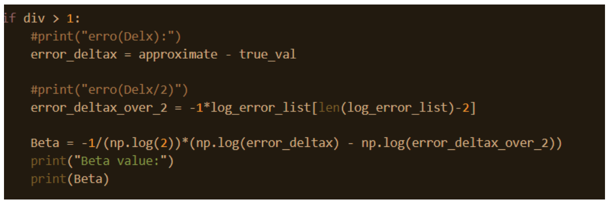




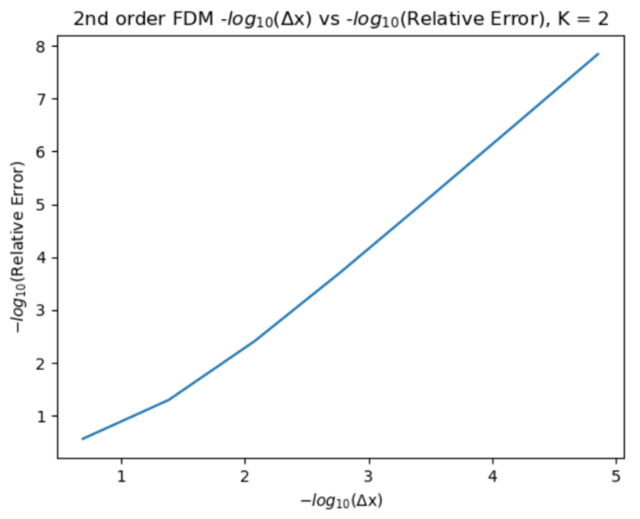
## 4. Convergence Rates and plots for 2nd order FDM:

Beta is measured using $\frac{-1}{\log(2)} \cdot \left(\log_{10} Error(\Delta x) - \log_{10} Error\left(\frac{\Delta x}{2}\right)\right)$, where $Error(\Delta x) = approximate - true.$

This code uses the index of the relative error array created to calculate Beta as follows:

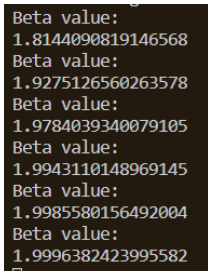


```
if div > 1:
    #print("erro(Delx):")
    error_deltax = approximate - true_val

    #print("erro(Delx/2)")
    error_deltax_over_2 = -1*log_error_list[len(log_error_list)-2]

    Beta = -1/(np.log(2))*(np.log(error_deltax) - np.log(error_deltax_over_2))
    print("Beta value:")
    print(Beta)
```



Slope is 1.8 with $R^2$= .993

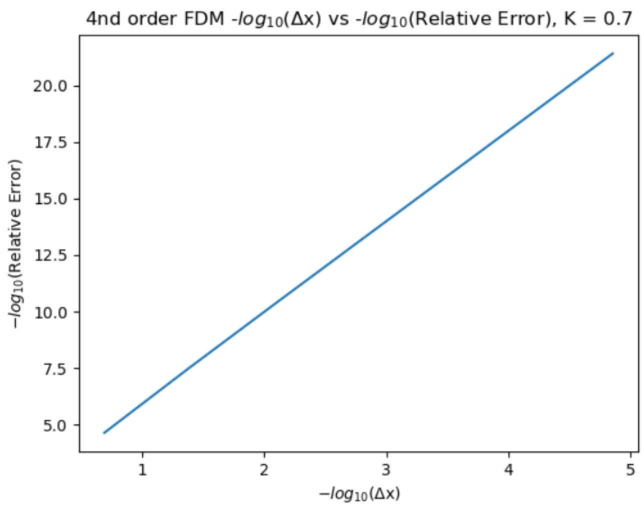
This output is the following in Python:



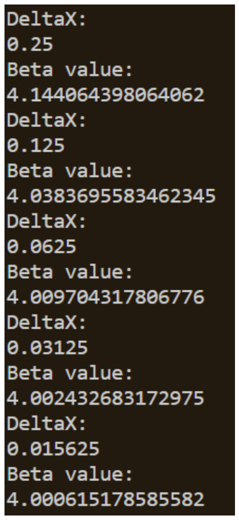
```
Beta value:
1.8144090819146568
Beta value:
1.9275126560263578
Beta value:
1.9784039340079105
Beta value:
1.9943110148969145
Beta value:
1.9985580156492004
Beta value:
1.9996382423995582
```

## 5. Convergence Rates and plots for 4th order FDM:

Slope approaches 4.44, with $R^2 = .99$



Beta values for 4th order FDM:



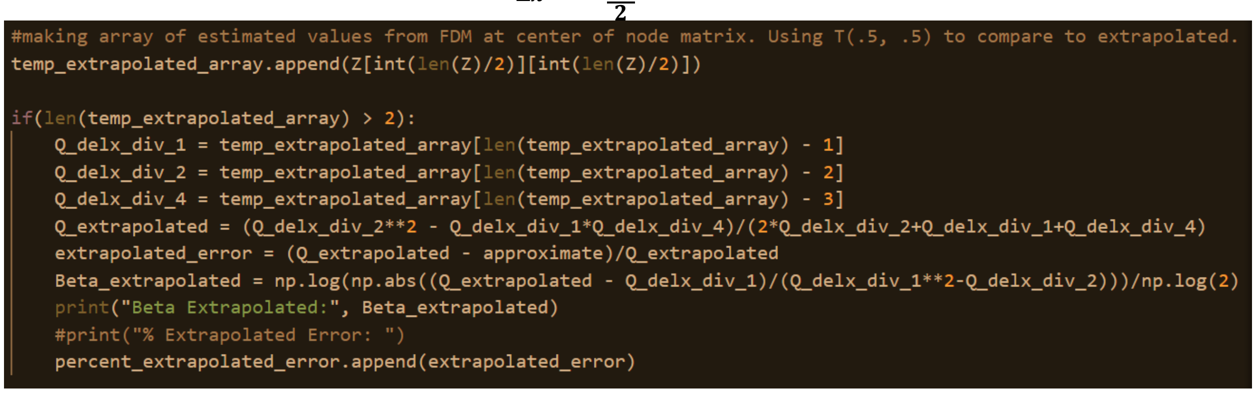
```
DeltaX:
0.25
Beta value:
4.144064398064062
DeltaX:
0.125
Beta value:
4.0383695583462345
DeltaX:
0.0625
Beta value:
4.009704317806776
DeltaX:
0.03125
Beta value:
4.002432683172975
DeltaX:
0.015625
Beta value:
4.000615178585582
```

## 6. Applying Richardson Extrapolation to FDM Output

$$\boldsymbol{Q_{extrapolated} = (Q^2_{\frac{\Delta x}{2}} - Q^1_{\Delta x} * Q^1_{\frac{\Delta x}{4}})/(\,2Q^1_{\frac{\Delta x}{2}} + Q^1_{\Delta x} + Q^1_{\frac{\Delta x}{4}})}$$

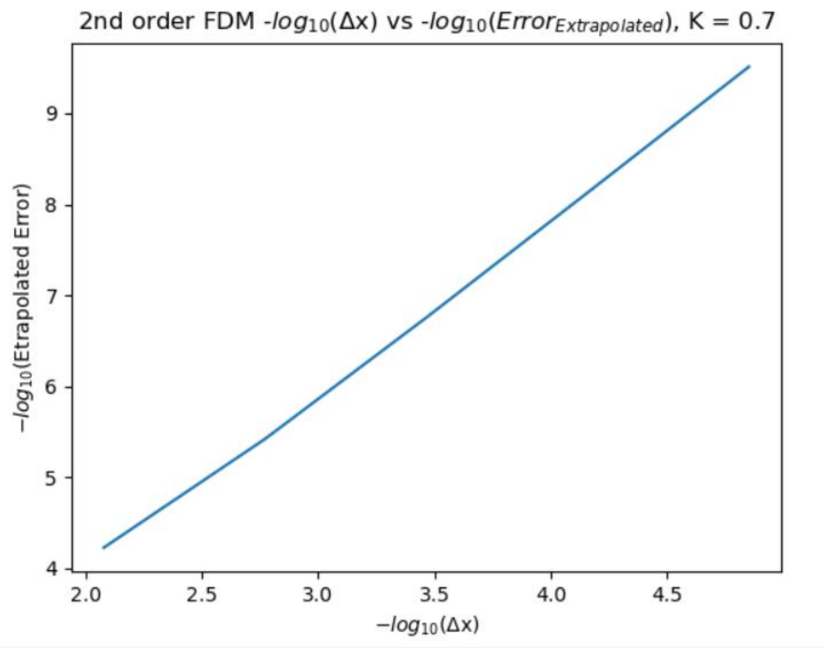
Each iteration of increasing step size, adds the FDM output at T(.5, .5) to the an array. Once this array has a length greater than 2, the Richardson extrapolation can be applied. This is useful in comparing the convergence of the FDM output to an extrapolated value. The convergence ($\beta$) is calculated by:

$$\frac{\boldsymbol{log(Q_{extrapolated} - Q_{\Delta x})}}{\boldsymbol{Q^2_{\Delta x} - Q_{\frac{\Delta x}{2}}}} * \frac{\boldsymbol{1}}{\boldsymbol{log}(2)}$$

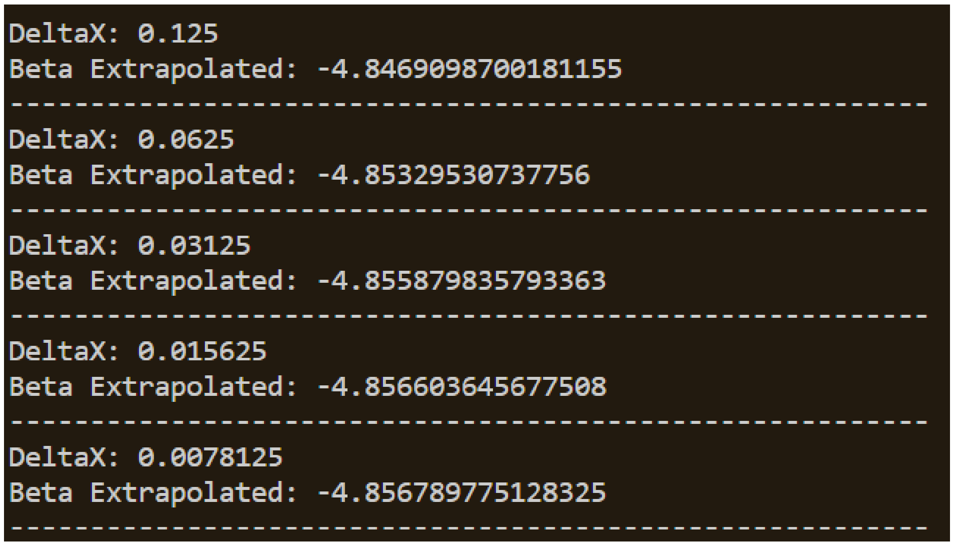


```
#making array of estimated values from FDM at center of node matrix. Using T(.5, .5) to compare to extrapolated.
temp_extrapolated_array.append(Z[int(len(Z)/2)][int(len(Z)/2)])

if(len(temp_extrapolated_array) > 2):
    Q_delx_div_1 = temp_extrapolated_array[len(temp_extrapolated_array) - 1]
    Q_delx_div_2 = temp_extrapolated_array[len(temp_extrapolated_array) - 2]
    Q_delx_div_4 = temp_extrapolated_array[len(temp_extrapolated_array) - 3]
    Q_extrapolated = (Q_delx_div_2**2 - Q_delx_div_1*Q_delx_div_4)/(2*Q_delx_div_2+Q_delx_div_1+Q_delx_div_4)
    extrapolated_error = (Q_extrapolated - approximate)/Q_extrapolated
    Beta_extrapolated = np.log(np.abs((Q_extrapolated - Q_delx_div_1)/(Q_delx_div_1**2-Q_delx_div_2)))/np.log(2)
    print("Beta Extrapolated:", Beta_extrapolated)
    #print("% Extrapolated Error: ")
    percent_extrapolated_error.append(extrapolated_error)
```

### 2nd order output log($\Delta x$) vs -log(Error from Extrapolation), K = .7:



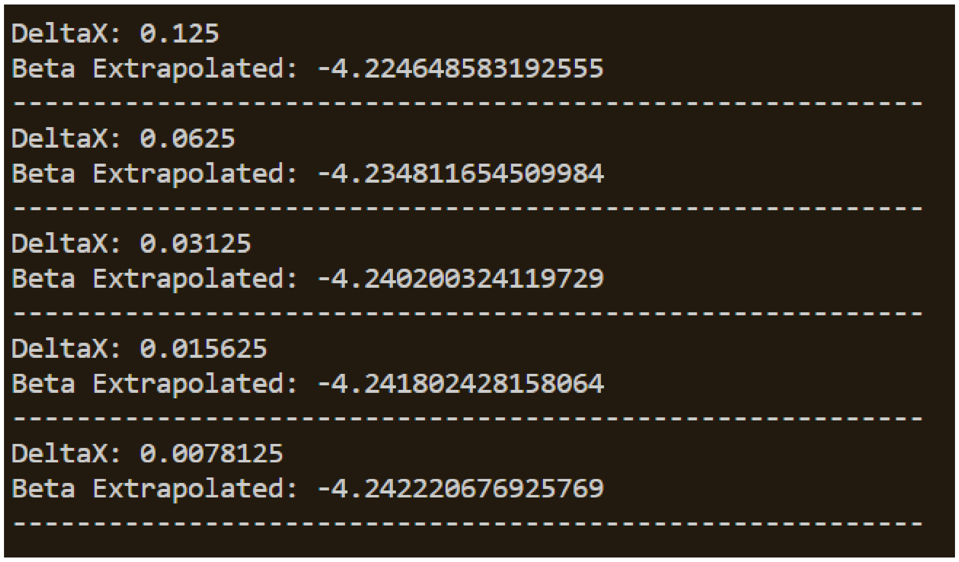
**2nd order Convergence values using Extrapolation, K = .7:**



```
DeltaX: 0.125
Beta Extrapolated: -4.8469098700181155
---------------------------------------------------------
DeltaX: 0.0625
Beta Extrapolated: -4.85329530737756
---------------------------------------------------------
DeltaX: 0.03125
Beta Extrapolated: -4.855879835793363
---------------------------------------------------------
DeltaX: 0.015625
Beta Extrapolated: -4.856603645677508
---------------------------------------------------------
DeltaX: 0.0078125
Beta Extrapolated: -4.856789775128325
---------------------------------------------------------
```

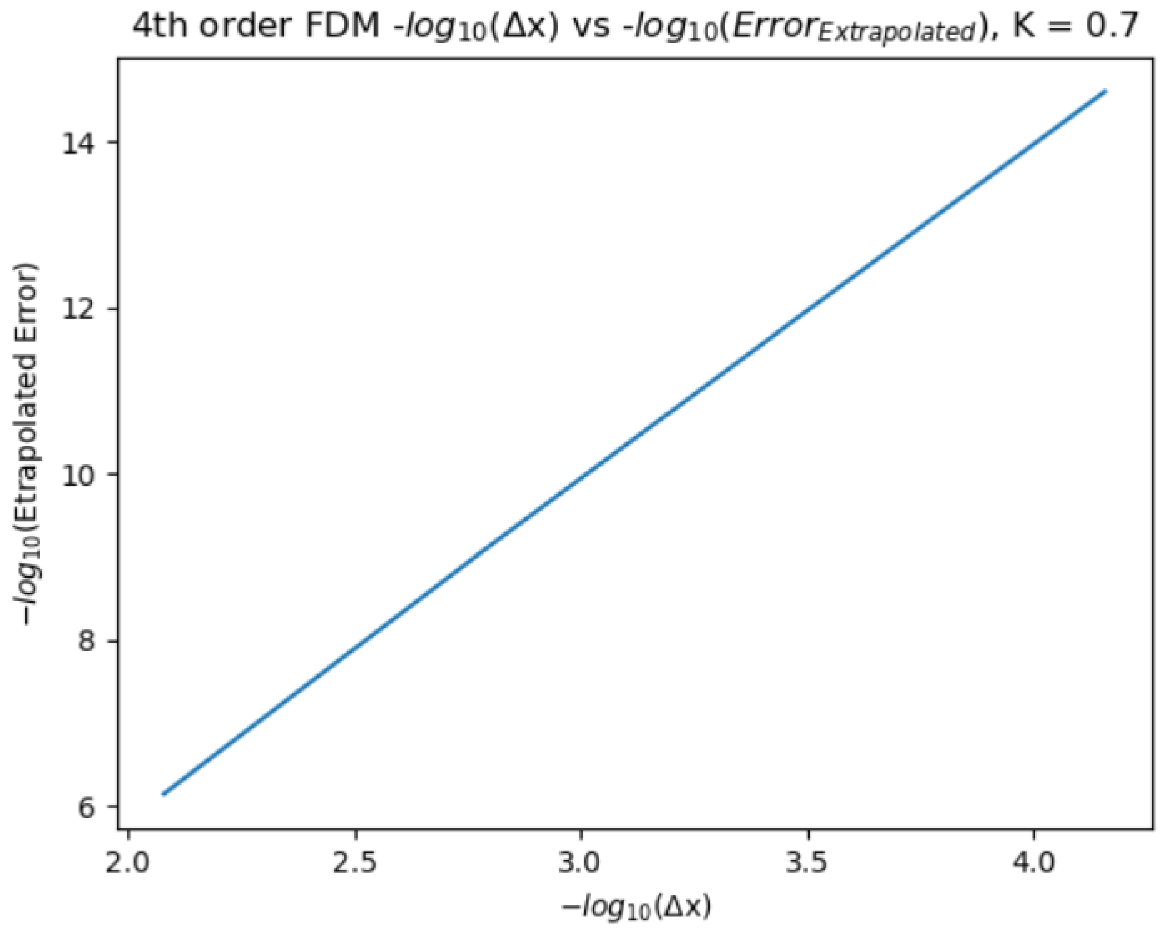
**2nd order Convergence values using Extrapolation, K = 1:**

***smaller convergence value can be observed when comparing to K = .7**

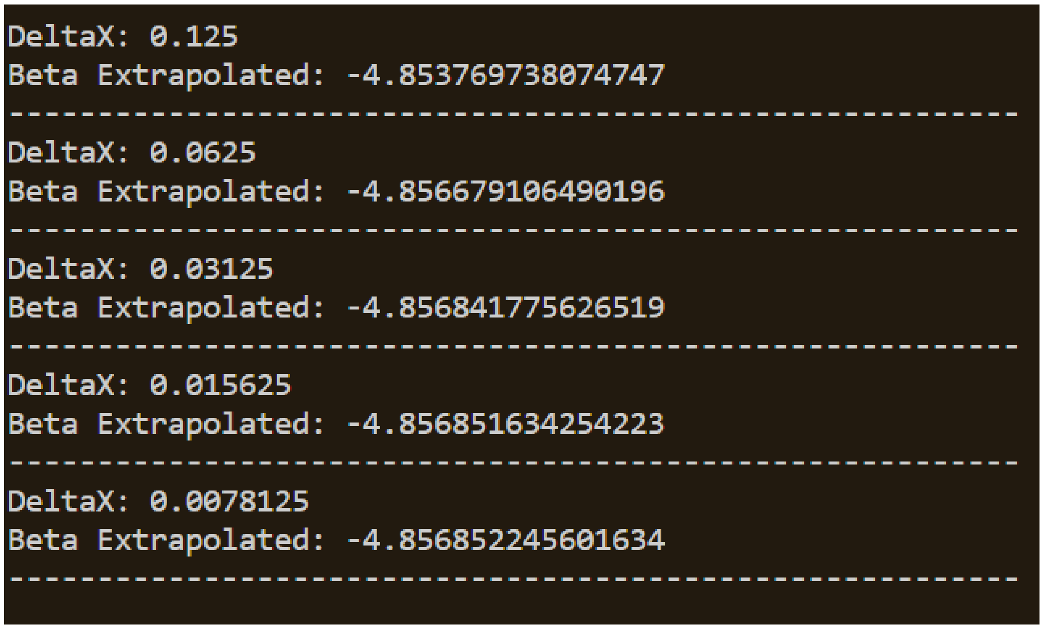


```
DeltaX: 0.125
Beta Extrapolated: -4.224648583192555
--------------------------------------------------------
DeltaX: 0.0625
Beta Extrapolated: -4.234811654509984
--------------------------------------------------------
DeltaX: 0.03125
Beta Extrapolated: -4.240200324119729
--------------------------------------------------------
DeltaX: 0.015625
Beta Extrapolated: -4.241802428158064
--------------------------------------------------------
DeltaX: 0.0078125
Beta Extrapolated: -4.242220676925769
--------------------------------------------------------
```

**4th order output log($\Delta x$) vs -log(Error from Extrapolation), K = .7:**



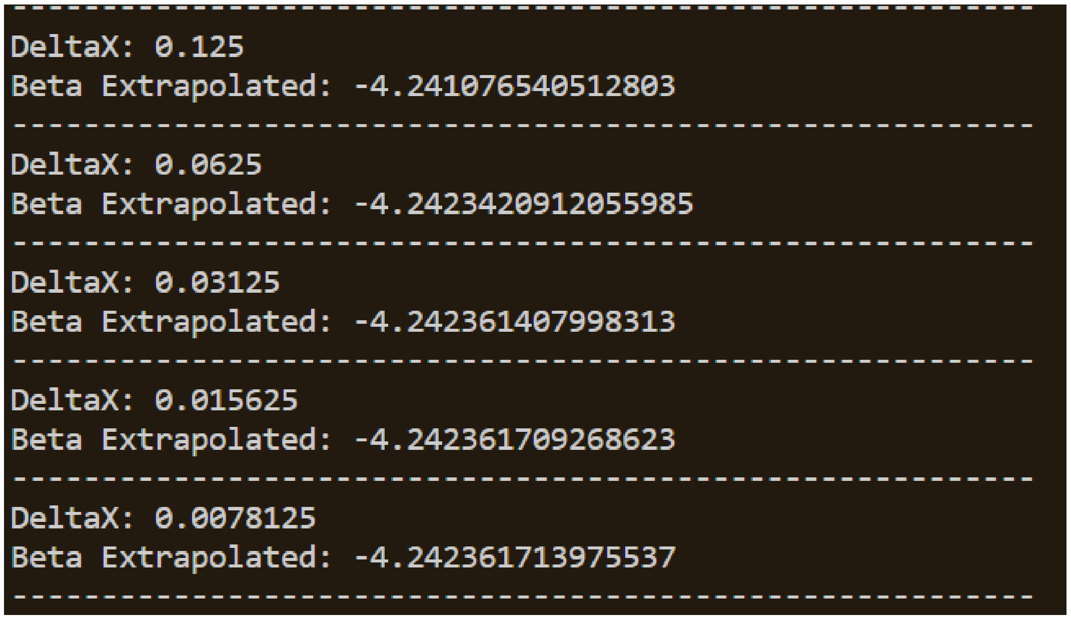
**4th order Convergence values using Extrapolation, K = .7:**



```
DeltaX: 0.125
Beta Extrapolated: -4.853769738074747
----------------------------------------------------------
DeltaX: 0.0625
Beta Extrapolated: -4.856679106490196
----------------------------------------------------------
DeltaX: 0.03125
Beta Extrapolated: -4.856841775626519
----------------------------------------------------------
DeltaX: 0.015625
Beta Extrapolated: -4.856851634254223
----------------------------------------------------------
DeltaX: 0.0078125
Beta Extrapolated: -4.856852245601634
----------------------------------------------------------
```

**4th order Convergence values using Extrapolation, K = 1:**

***smaller convergence value can be observed when comparing to K = .7**



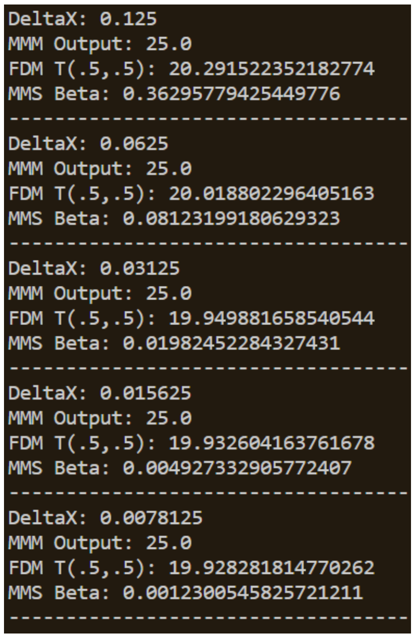
```
----------------------------------------------------------
DeltaX: 0.125
Beta Extrapolated: -4.241076540512803
----------------------------------------------------------
DeltaX: 0.0625
Beta Extrapolated: -4.2423420912055985
----------------------------------------------------------
DeltaX: 0.03125
Beta Extrapolated: -4.242361407998313
----------------------------------------------------------
DeltaX: 0.015625
Beta Extrapolated: -4.242361709268623
----------------------------------------------------------
DeltaX: 0.0078125
Beta Extrapolated: -4.242361713975537
----------------------------------------------------------
```

## 7. Convergence of Estimated Heat Distribution Using Method of Manufactured Solutions:

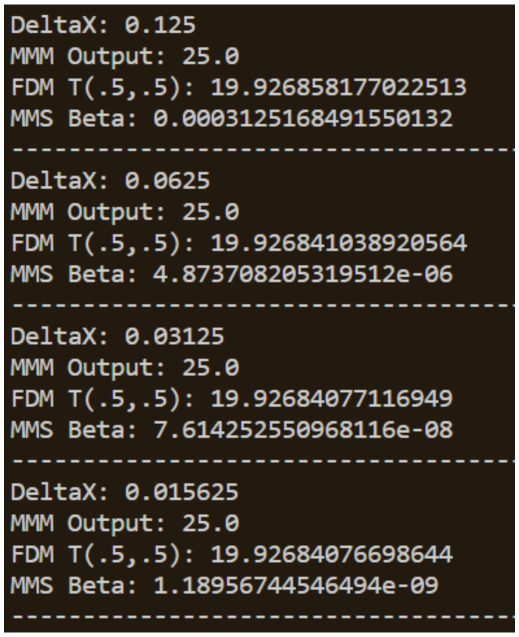
The set of boundary conditions could be approximated by the solution: T = 100*$y^2$ at x = .5 cm.

This manufactured solution can be compared to the FDM output at T(.5, .5).

The following convergence values were obtained used this Method of Manufactured Solutions for 2nd and 4th order FDM comparison, K = .7:



```
DeltaX: 0.125
MMM Output: 25.0
FDM T(.5,.5): 20.291522352182774
MMS Beta: 0.36295779425449776
----------------------------------
DeltaX: 0.0625
MMM Output: 25.0
FDM T(.5,.5): 20.018802296405163
MMS Beta: 0.08123199180629323
----------------------------------
DeltaX: 0.03125
MMM Output: 25.0
FDM T(.5,.5): 19.949881658540544
MMS Beta: 0.01982452284327431
----------------------------------
DeltaX: 0.015625
MMM Output: 25.0
FDM T(.5,.5): 19.932604163761678
MMS Beta: 0.004927332905772407
----------------------------------
DeltaX: 0.0078125
MMM Output: 25.0
FDM T(.5,.5): 19.928281814770262
MMS Beta: 0.0012300545825721211
----------------------------------
```



```
DeltaX: 0.125
MMM Output: 25.0
FDM T(.5,.5): 19.926858177022513
MMS Beta: 0.0003125168491550132
-----------------------------------
DeltaX: 0.0625
MMM Output: 25.0
FDM T(.5,.5): 19.926841038920564
MMS Beta: 4.873708205319512e-06
-----------------------------------
DeltaX: 0.03125
MMM Output: 25.0
FDM T(.5,.5): 19.92684077116949
MMS Beta: 7.614252550968116e-08
-----------------------------------
DeltaX: 0.015625
MMM Output: 25.0
FDM T(.5,.5): 19.92684076698644
MMS Beta: 1.18956744546494e-09
-----------------------------------
```

## 8. Approximating Heat Flux Across Top Surface Using 1/3 Simpson Integration:

Analytical Solution to Total Heat Flux:

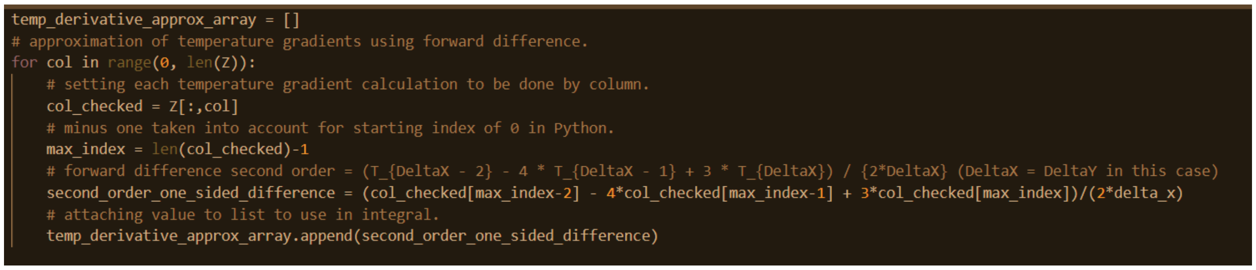
$\dot{q} = \int_0^x -k \cdot thickness \cdot \frac{dT}{dy} \cdot dx$, where $\frac{dT}{dy} = \frac{d}{dy}(\frac{100\sinh(K\pi y)}{\sinh(K\pi)} \cdot sin(\pi x))$

This leads to $\dot{q} = -200 \cdot \frac{K}{\tanh(\pi)}$. For K = 1, $\dot{q} =$ -200.748374639

Using finite difference, this can be approximated using the forward finite difference.

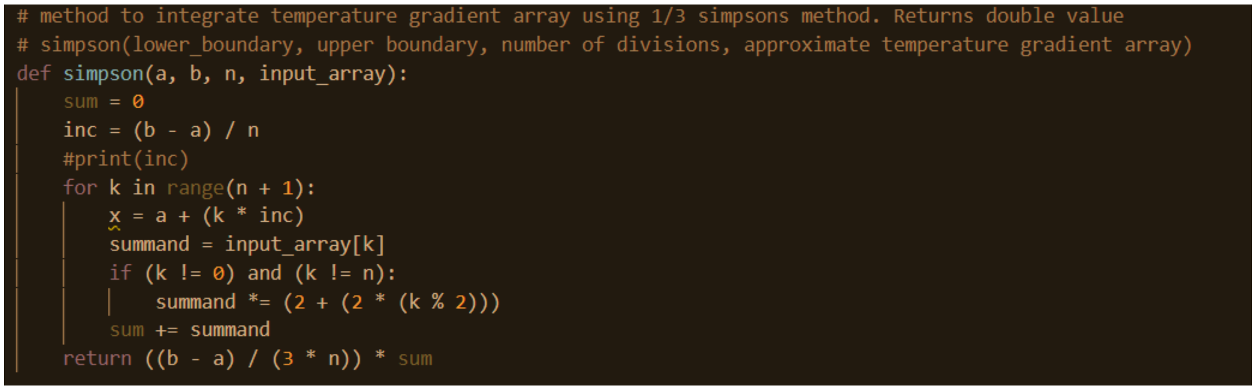
For second order FDM, $\frac{dT}{dy} \approx \frac{T_{(i,j_{max}-2)} - 4 \cdot T_{(i,j_{max}-1)} + 3 \cdot T_{(i,j_{\max})}}{2 \cdot \Delta x}$

For fourth order FDM, $\frac{dT}{dy} \approx \frac{-2 \cdot T_{(i,j_{max}-3)} + 9T_{(i,j_{max}-2)} - 18 \cdot T_{(i,j_{max}-1)} + 11 \cdot T_{(i,j_{\max})}}{6 \cdot \Delta x}$



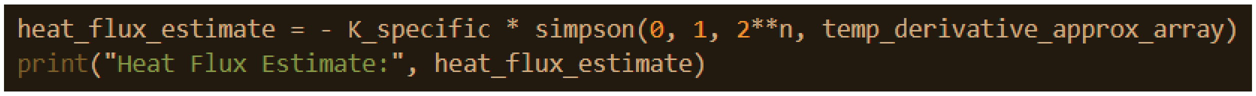
```
temp_derivative_approx_array = []
# approximation of temperature gradients using forward difference.
for col in range(0, len(Z)):
    # setting each temperature gradient calculation to be done by column.
    col_checked = Z[:,col]
    # minus one taken into account for starting index of 0 in Python.
    max_index = len(col_checked)-1
    # forward difference second order = (T_{DeltaX - 2} - 4 * T_{DeltaX - 1} + 3 * T_{DeltaX}) / {2*DeltaX} (DeltaX = DeltaY in this case)
    second_order_one_sided_difference = (col_checked[max_index-2] - 4*col_checked[max_index-1] + 3*col_checked[max_index])/(2*delta_x)
    # attaching value to list to use in integral.
    temp_derivative_approx_array.append(second_order_one_sided_difference)
```

This is then integrated using the following 1/3 Simpsons method:



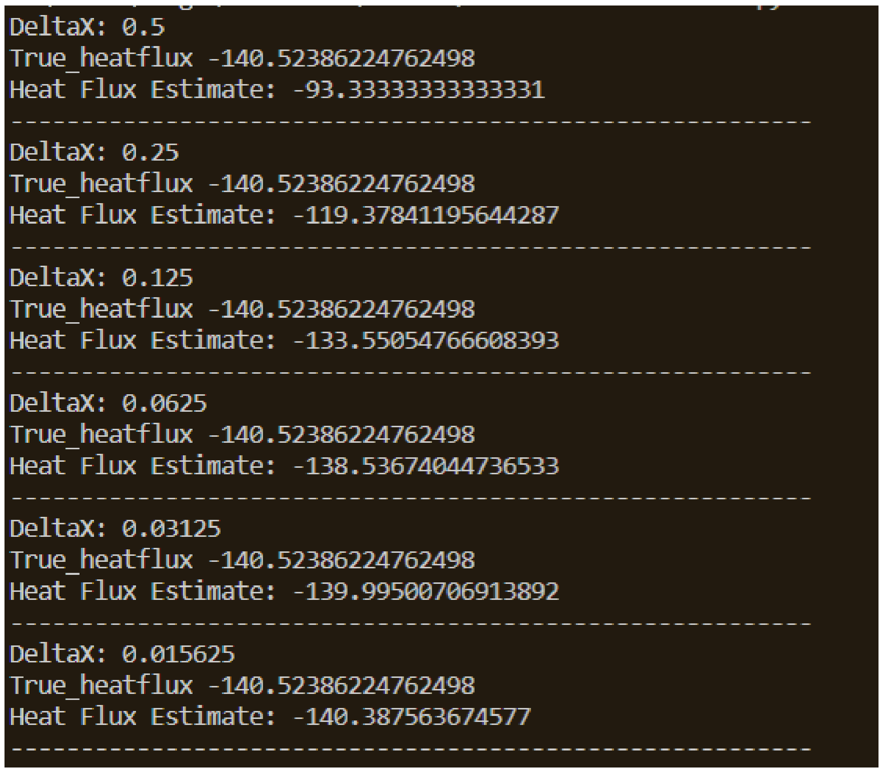
```
# method to integrate temperature gradient array using 1/3 simpsons method. Returns double value
# simpson(lower_boundary, upper boundary, number of divisions, approximate temperature gradient array)
def simpson(a, b, n, input_array):
    sum = 0
    inc = (b - a) / n
    #print(inc)
    for k in range(n + 1):
        x = a + (k * inc)
        summand = input_array[k]
        if (k != 0) and (k != n):
            summand *= (2 + (2 * (k % 2)))
        sum += summand
    return ((b - a) / (3 * n)) * sum
```

The estimated total heat flux is then printed using the code below:



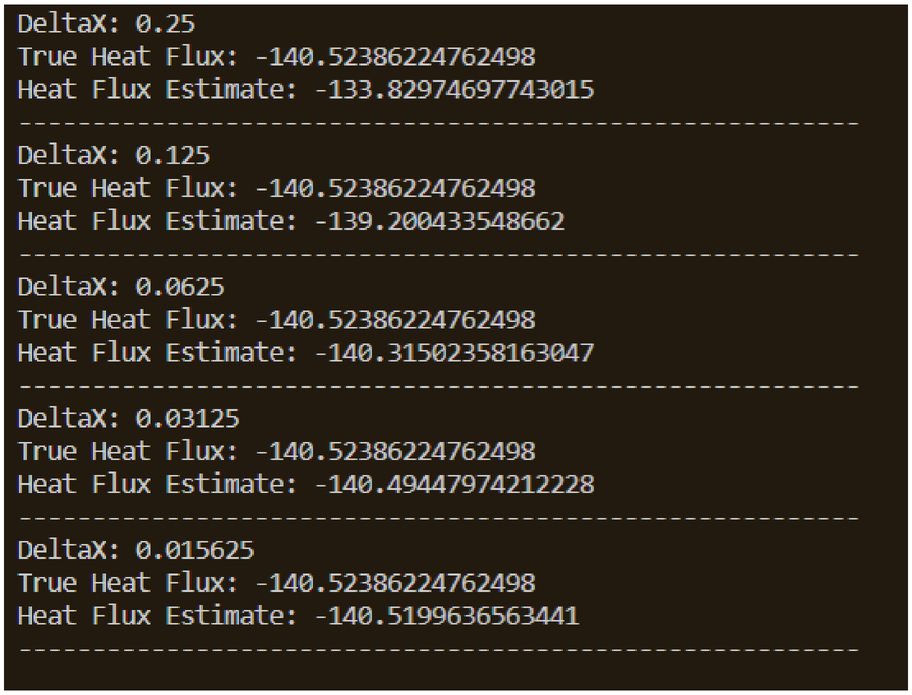
```
heat_flux_estimate = - K_specific * simpson(0, 1, 2**n, temp_derivative_approx_array)
print("Heat Flux Estimate:", heat_flux_estimate)
```

2nd order estimated heat flux from K = .7



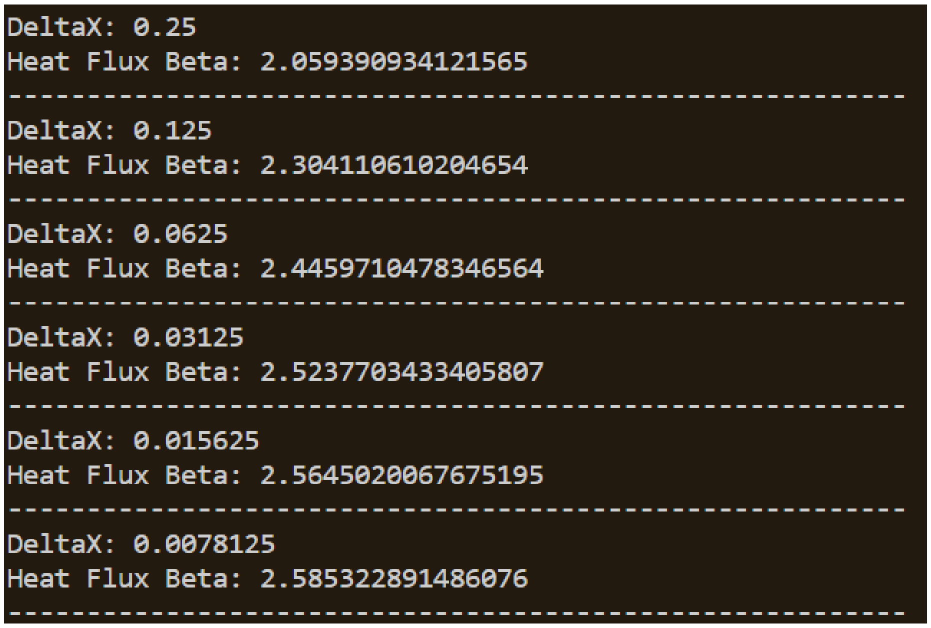
```
DeltaX: 0.5
True_heatflux -140.52386224762498
Heat Flux Estimate: -93.33333333333331
---------------------------------------------------------
DeltaX: 0.25
True_heatflux -140.52386224762498
Heat Flux Estimate: -119.3784119564287
---------------------------------------------------------
DeltaX: 0.125
True_heatflux -140.52386224762498
Heat Flux Estimate: -133.55054766608393
---------------------------------------------------------
DeltaX: 0.0625
True_heatflux -140.52386224762498
Heat Flux Estimate: -138.53674044736533
---------------------------------------------------------
DeltaX: 0.03125
True_heatflux -140.52386224762498
Heat Flux Estimate: -139.99500706913892
---------------------------------------------------------
DeltaX: 0.015625
True_heatflux -140.52386224762498
Heat Flux Estimate: -140.387563674577
---------------------------------------------------------
```

4th order estimated heat flux, K = .7:



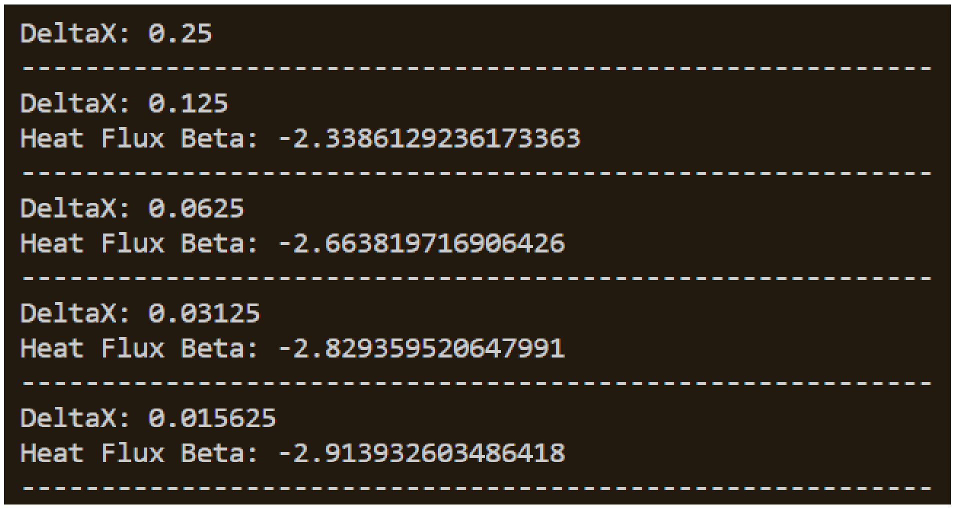
```
DeltaX: 0.25
True Heat Flux: -140.52386224762498
Heat Flux Estimate: -133.82974697743015
---------------------------------------------------------
DeltaX: 0.125
True Heat Flux: -140.52386224762498
Heat Flux Estimate: -139.200433548662
---------------------------------------------------------
DeltaX: 0.0625
True Heat Flux: -140.52386224762498
Heat Flux Estimate: -140.31502358163047
---------------------------------------------------------
DeltaX: 0.03125
True Heat Flux: -140.52386224762498
Heat Flux Estimate: -140.49447974212228
---------------------------------------------------------
DeltaX: 0.015625
True Heat Flux: -140.52386224762498
Heat Flux Estimate: -140.5199636563441
---------------------------------------------------------
```

2nd Order Heat Flux Convergence, K = .7:



```
DeltaX: 0.25
Heat Flux Beta: 2.059390934121565
----------------------------------------------------------
DeltaX: 0.125
Heat Flux Beta: 2.304110610204654
----------------------------------------------------------
DeltaX: 0.0625
Heat Flux Beta: 2.4459710478346564
----------------------------------------------------------
DeltaX: 0.03125
Heat Flux Beta: 2.5237703433405807
----------------------------------------------------------
DeltaX: 0.015625
Heat Flux Beta: 2.5645020067675195
----------------------------------------------------------
DeltaX: 0.0078125
Heat Flux Beta: 2.585322891486076
----------------------------------------------------------
```

4th Order Heat Flux Convergence, K = .7:



```
DeltaX: 0.25
----------------------------------------------------------
DeltaX: 0.125
Heat Flux Beta: -2.3386129236173363
----------------------------------------------------------
DeltaX: 0.0625
Heat Flux Beta: -2.663819716906426
----------------------------------------------------------
DeltaX: 0.03125
Heat Flux Beta: -2.829359520647991
----------------------------------------------------------
DeltaX: 0.015625
Heat Flux Beta: -2.913932603486418
----------------------------------------------------------
```

## Conclusion:

This project made use of FDM to break down a boundary condition problem into a set of finite difference equations. This set of equations was then broken down into an assembled matrix and resultant matrix from the boundary conditions. These were then solved with linear algebra to get a matrix of temperatures representing the distribution on the surface defined.

Python was used out of convenience, however there are issues with accuracy of Python, as the computer would shut down when attempting to solve for 4th order meshes with degrees greater than 8. Extended precision packages would come in use to increase accuracy. Increasing efficiency may also be changed by avoiding the use of available Python packages for linear algebra. A programming language such as C++ might increase available size of matrices to solve for temperature distributions.

The estimated heat temperatures converged when compared to the True analytical result and converged when compared to the Method of Manufactured Solutions.

When comparing 2nd order to 4th order convergence, 4th order converged much faster than 2nd order for the same K and DeltaX values.

This assignment was useful in seeing how quickly the accuracy can improve with higher orders of FDM convergences for solutions to this set of partial differential equations along with seeing the effect of Richardson Extrapolation and Method of Manufactured Solutions to check convergence rates.