Python code for One dimensional heat equation

```
#we will first transform the one dimensional heat equation using the finite difference for
#then we impose start and end boundary condition values on the equtaion
#finally we plot the changes in temperature per distance
```

```
#Note
#(1) that we are told to heat the bar to a fixed positive temperatures at both ends
#this will be given by setting the left and right boundary conditions to a positive values


#(2) we need to also set the initial condition to a non-trivial temperature distribution-t
#temperature to a non-zero (non-trivial) => u(0,x)=f(x)

#(3) we need to set the heat flux approximation in the finite difference scheme
#this we will approximate it as follows
#Note in the code below==heat flux approximation in finite difference scheme will be denot
```

$$=> heat\ flux\ defined\ :\ \overrightarrow{q} = -k\nabla U / \Delta x$$

$$=> \rho c_p \Delta x \frac{\mathrm{dU}}{\mathrm{dt}} \approx \frac{-k\nabla U(\nabla)}{\Delta x}$$

$$=> In\ finite\ difference\ we\ have :$$

$$=> \frac{dU}{dt} = \frac{k}{\rho c_p}\left(\frac{U_{i-1}+U_{i+1}-2U_i}{\Delta x^2}\right)$$

$$=> We\ approximate\ the\ heat\ flux\ in\ finite\ difference\ as :$$

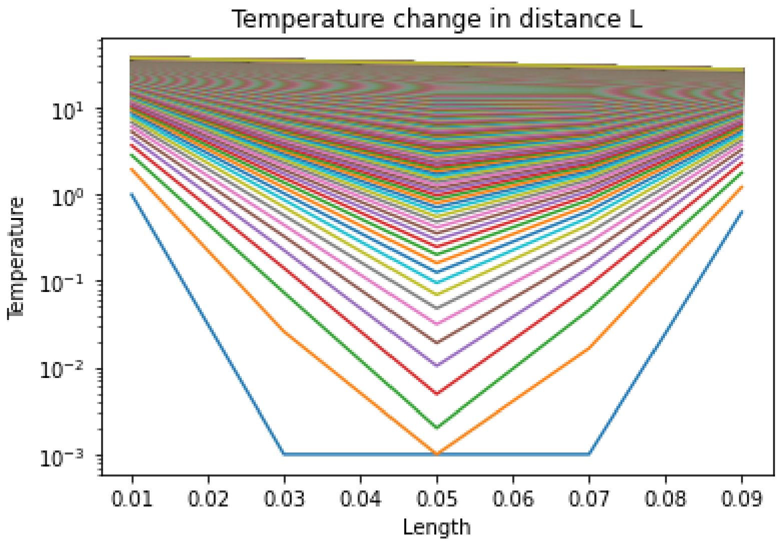
$$=> \frac{dU}{dt} = \alpha\left(\frac{U_{i-1}+U_{i+1}-2U_i}{\Delta x^2}\right)$$

$$=> \alpha = \frac{k}{\rho c_p} = 0.0001$$

```
#we will first transform the one dimensional heat equation using the finite difference for
#then we impose start and end boundary condition values on the equtaion
#finally we plot the changes in temperature per distance

#importing important libraries
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")


#setting a random alpha value for the heat equation
cons=0.0001 #NOTE ,this can be changed accordingly based on the heat equation formulation

s=5  # x number of steps
T0=0 # setting initial condition temperature to start from zero
```

```
L=0.1  # critical length value
dx=L/s # change in step length
final_temp=100 #setting final temperature
dt=0.1 # time step


t=np.arange(0,final_temp,dt)
x=np.linspace(dx/2,L-dx/2,s) # setting x in the specified interval

T=np.ones(s)*T0 #converting initial Temp condition from scalar to vector
U=np.empty(s) #derivertive initialization


left_end_temp=20
right_end_temp=0

for _ in range(1,len(t)):
  # applying left  boundary condition
  U[0]=0
  U[0]=((T[1]-T[0])/dx**2-(T[0]-left_end_temp)/dx**2)*cons
  for k in range(1,s-1):

    #the finite difference transformation of the one dimensional heat Pde equation
    U[k]=((T[k+1]-T[k])/dx**2-(T[k]-T[k-1])/dx**2)*cons

  # applying the right boundary condition
  U[s-1]=0
  U[s-1]=((right_end_temp-T[s-1])/dx**2-(T[s-1]-T[s-2])/dx**2)*cons




  #plotting the results
  T+=U*dt
  plt.semilogy(x,T)
  plt.title('Temperature change in distance L')
  plt.ylabel('Temperature')
  plt.xlabel('Length')
  # plt.legend()
```


```
#importing important libraries
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")


#setting a random alpha value for the heat equation
cons =0.0001 #  (alpha) #NOTE ,this can be changed accordingly based on the heat equation
t=0.001

s=5  # x number of steps
T0=np.sin(t) # setting initial condition temperature to start from non-zero (non trival va

L=0.1  # critical length value
dx=L/s # change in step length
final_temp=100 #setting final temperature
dt=0.1 # time step
left_end_temp=40
right_end_temp=25


t=np.arange(0,final_temp,dt)
x=np.linspace(dx/2,L-dx/2,s) # setting x in the specified interval

T=np.ones(s)*T0 #converting initial Temp condition from scalar to vector
U=np.empty(s) #derivertive initialization


for _ in range(1,len(t)):
  # applying the left U[0,k]=0 boundary condition
  U[0]=0 # left boundary condition
  U[0]=((T[1]-T[0])/dx**2-(T[0]-left_end_temp)/dx**2)*cons
  for k in range(1,s-1):

    #the finite difference transformation of the one dimensional heat Pde equation
    U[k]=((T[k+1]-T[k])/dx**2-(T[k]-T[k-1])/dx**2)*cons

  #applying the U[s-1,k] = U[s-3,k] # right neumann boundary condition
  U[s-1]=U[s-3] # newman right boundary condition
  U[s-1]=((right_end_temp-T[s-1])/dx**2-(T[s-1]-T[s-2])/dx**2)*cons



  #plotting the results
  T+=U*dt
  plt.semilogy(x,T)
  plt.title('Temperature change in distance L')
  plt.ylabel('Temperature')
  plt.xlabel('Length')
  # plt.legend()
```


Temperature change in distance L
Temperature
Length
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
10^1
10^0
10^{-1}
10^{-2}
10^{-3}