```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 27 23:54:09 2019
HW5a
@author: Ryan Dalby
"""
import numpy as np
import matplotlib.pyplot as plt
from NaiveGaussElimination import gaussElimination

def find_temperature_profile_finite(firstx, firstT, secondx, secondT, numberOfNodes, shouldPlot = F
    """
    Given two boundary conditions (x1, T1) and (x2, T2), and number of nodes(nodes are inclusive of
    and an xDesired where the temperature will be calcuated at 
    Will display a plot of T vs x(if indicated to do so) and return np arrays of the x values and t
    """
    d = 0.1 #m
    h = 20 #W/m^2*K
    k = 200 #W/m * K
    Too = 300 #K surrounding temperature, time at t = infinity
    P = np.pi * d
    A = np.pi * (d / 2) ** 2

    deltaX = (secondx-firstx)/(numberOfNodes-1) #delta x
    alpha = (h * P) / (k * A)

    #create A matrix that represents our finite difference equations reduced from numberOfNodes to
    n = numberOfNodes - 2 #The number of equations needed is numberofNodes - 2 since we know the in
    diagNum = (2 + alpha * deltaX**2)
    A = np.diag(np.full((n),diagNum)) + np.diag(np.full((n-1),-1), 1) + np.diag(np.full((n-1),-1),
    #Create b vector of size n (subtract values from first and second since we know 2 of the nodes
    diagb = Too * alpha * deltaX**2
    b = np.full((n,1), diagb)
    b[0] += firstT
    b[n-1] += secondT 
    TVals = np.concatenate([[firstT], np.squeeze(gaussElimination(A,b)), [secondT]])
    xVals = np.linspace(firstx, secondx, numberOfNodes)
    if(shouldPlot):
        fig, ax = plt.subplots()
        ax.plot(xVals, TVals)
        ax.set_title("Finite Difference Method temperature vs x solution")
        ax.set_xlabel("x(m)")
        ax.set_ylabel("Temperature(K)")

    return xVals, TVals


finitex, finiteT = find_temperature_profile_finite(0, 600, 2.0, 350, 51, shouldPlot=True)
```