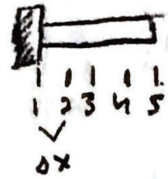
ME EN 2450 | Ryan Dalby | Homework 5

Exercise 4

$$\frac{d^2T}{dx^2} = \frac{hP}{KA}(T - T_\infty)$$ From exercise 2

$$\frac{d^2T}{dx^2} - \frac{hP}{KA}(T_i - T_\infty) = 0$$



$L = 2.0m$

$\Delta x = \frac{2.0}{4} = 0.5$

$$\frac{d^2T}{dx^2} - 4(T_i - 300) = 0$$

$$\frac{hP}{KA} = \frac{(20)(\pi(1))}{(200)\pi(\frac{1}{2})^2} = 4$$

$$\frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} - 4(T_i - 300) = 0$$

$$-T_{i-1} + (2 + 4\overset{0.5}{\Delta x^2})T_i - T_{i+1} = 4(300)\Delta x^2$$

$$-T_1 + 3T_2 - T_3 = 300$$
$$-T_2 + 3T_3 - T_4 = 300$$
$$-T_3 + 3T_4 - T_5 = 300$$

when $T_1 = 600K$, $T_5 = 350K$

$$\begin{bmatrix} 3 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 900 \\ 300 \\ 650 \end{bmatrix}$$

Forward Elimination

$$\left[\begin{array}{ccc|c} 3 & -1 & 0 & 900 \\ 0 & 2.67 & -1 & 600 \\ 0 & -1 & 3 & 650 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 3 & -1 & 0 & 900 \\ 0 & 2.67 & -1 & 600 \\ 0 & 0 & 2.625 & 875 \end{array}\right]$$

Back Substitution

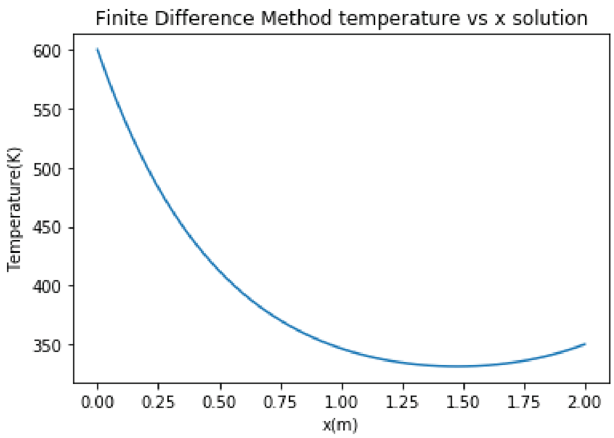
$$\begin{bmatrix} T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 416.67 \\ 350 \\ 333.33 \end{bmatrix}$$

$$\boxed{\begin{aligned} T_1 &= 600K \\ T_2 &= 416.67K \\ T_3 &= 350K \\ T_4 &= 333.33K \\ T_5 &= 350K \end{aligned}}$$

```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW5/HW5a.py', wdir='C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/
HW5')
```




```
In [2]:
```

```python
# -*- 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 i
    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)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 26 21:13:09 2019
Naive Gauss Elimination from HW3 Exercise 3
@author: Ryan Dalby
"""

import numpy as np

def gaussElimination(A, b):
    """
    Uses naive gauss elimination method without pivoting to determine solutions to Ax = b
    Will directly modify A passed into upper triangular form
    """
    n = np.shape(A)[0] #assuming nxn matrix

#    print("Naive Gauss Elimination Steps:\n")
#    print("Original Matrix:")
#    print(A,"\n")
#    print("Perform forward elimination:")
    #Perform forward elimination
    for k in range(n - 1):
        for i in range(k + 1, n):
            s = A[i,k] / A[k,k]
            for j in range(k, n):
                A[i,j] = A[i,j] - s * A[k,j]
#                print("A = \n{}\nb = \n{}\n".format(A,b))
            b[i] = b[i] - s * b[k]
#            print("A = \n{}\nb = \n{}\n".format(A,b))


#    print()
#    print("Perform back substitution:")
    #Perform back substitution
    xSol = np.zeros(shape = (n,1), dtype='float')
    xSol[n-1] = b[n-1] / A[n-1,n-1] #index last element of xSol and set it to the solution value fo
#    print("A = \n{}\nb = \n{}\n".format(A,b))
    for i in range(n-1, -1, -1):
        s = 0.0
        for j in range(i+1, n):
            s = s + A[i,j] * xSol[j]
        xSol[i] = (b[i] - s) / A[i,i]
#        print("x = \n{}\n".format(xSol))
    return xSol
```