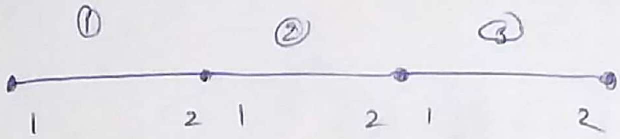
Assignment 3

Aakash
ME16B001

1. $$-\frac{d^2u}{dx^2} - u + x^2 = 0 \qquad 0 < x < 1 \qquad u(0) = u(1)$$

$$a = 1 \qquad c = -1 \qquad f = -x^2$$

(a) linear elements



$$Ku = f + Q$$

coefficient matrix — source vector

$$K\,\text{matrix} = \begin{bmatrix} K_{11}^1 & K_{12}^1 & 0 & 0 \\ K_{21}^1 & K_{22}^1 + K_{11}^2 & K_{12}^2 & 0 \\ 0 & K_{21}^2 & K_{22}^2 + K_{11}^3 & K_{12}^3 \\ 0 & 0 & K_{21}^3 & K_{22}^3 \end{bmatrix} \qquad f\,\text{matrix} = \begin{bmatrix} F_1^1 \\ F_1^2 + F_2^1 \\ F_2^2 + F_3^1 \\ F_3^2 \end{bmatrix}$$

$$K_{ij} = \int_{x_a}^{x_b} \left(a \frac{d\psi_i}{dx} \frac{d\psi_j}{dx} + c\,\psi_i \psi_j \right) dx \qquad F_i^e = \int_{x_a}^{x_b} f \psi_i^e \, dx$$

where $$\psi_1 = \frac{x_2 - x}{x_2 - x_1} = 1 - 3x \qquad \psi_2 = \frac{x - x_1}{x_2 - x_1} = 3x$$

Using the integral relation get

$$K = \begin{bmatrix} 2.888 & -3.0555 & 0 & 0 \\ -3.0555 & 5.777 & -3.0555 & 0 \\ 0 & -3.0555 & 5.777 & -3.0555 \\ 0 & 0 & -3.0555 & 2.888 \end{bmatrix}$$

$$f_{matrix} = \begin{bmatrix} -0.0030864 \\ -0.0432098 \\ -0.15432 \\ -0.13271 \end{bmatrix}$$

$$Q \text{ matrix} = \begin{bmatrix} m \\ 0 \\ 0 \\ n \end{bmatrix}$$

$KU = f + Q$

$$\begin{bmatrix} 2.888 & -3.0555 & 0 & 0 \\ -3.0555 & 5.777 & -3.0555 & 0 \\ 0 & -3.0555 & 5.777 & -3.0555 \\ 0 & 0 & -3.0555 & 2.888 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} -0.0030864 + m \\ -0.0432098 \\ -0.15432 \\ -0.13271 + n \end{bmatrix}$$

Boundary condition $u(0) = u(1)$

$\therefore$ let $u_1 = u_2 = p$ (some known quantity)

$$\left(2.888 + (-3.0555) + 0 + 0\right) u_1 = -0.0030864 + m$$

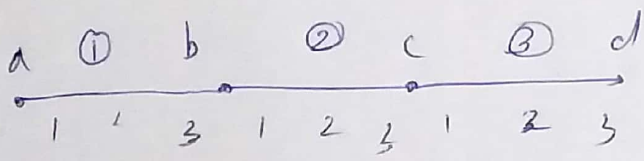
$$m = -\frac{67}{400} p + 0.030864$$

$$\left(0 + 0 + (-3.055) + 2.88\right) u_2 = -0.13271 + n$$

$$n = \frac{-67}{400} p + 0.13271$$

m & n can be substituted back in eqn. $KU = f + Q$ and solved.

(b) <u>Quadratic elements.</u>



$$\psi_1 = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}$$

$$\psi_2 = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}$$

$$\psi_3 = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

$$
\begin{bmatrix}
K_{11}^{1} & K_{12}^{1} & K_{13}^{1} & & & & \\
K_{21}^{1} & K_{22}^{1} & K_{23}^{1} & & & & \\
K_{31}^{1} & K_{32}^{1} & K_{33}^{1}+K_{11}^{2} & K_{12}^{2} & K_{13}^{2} & & \\
 & & K_{21}^{2} & K_{22}^{2} & K_{23}^{2} & & \\
 & & K_{31}^{2} & K_{32}^{2} & K_{33}^{2}+K_{11}^{3} & K_{12}^{3} & K_{13} \\
 & & & & K_{21}^{3} & K_{22}^{3} & K_{23}^{3} \\
 & & & & K_{31}^{3} & K_{32}^{3} & K_{33}^{3}
\end{bmatrix}
\begin{bmatrix}
U_a^{1} \\ U_a^{2} \\ U_b \\ U_b^{2} \\ U_c \\ U_c^{2} \\ U_d
\end{bmatrix}
=
\begin{bmatrix}
b_1^{1} \\ b_1^{2} \\ b_1^{3} + b_2^{1} \\ b_2^{2} \\ b_2^{3} + b_3^{1} \\ b_3^{2} \\ b_3^{3}
\end{bmatrix}
$$

In [1]:

```
# Problem (2)
# Part (a)

from sympy.solvers import solve
from sympy import Symbol
from sympy import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import *
from numpy.linalg import inv
from array import *
from scipy import linalg
x=Symbol('x')

# function to find k and f for linear elements
def kfmatixLin(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = (xb[i]-x)/(xb[i]-xa[i])
        psi_2 = (x-xa[i])/(xb[i]-xa[i])
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    for i in range(0,len(Ftemp)-2,2):
        F.append(Ftemp[i+1]+Ftemp[i+2])
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # two diagonals of the tridiagonal matrix
    diagA=[]
    diagB=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
    # print('diagA',diagA)
    # NOTE: no need for diagC as it will always be same as diagA
```

```
    diagB.append(k[0][0])
    for i in range(nEle-1):
        diagB.append(k[i][3]+k[i+1][0])
    diagB.append(k[nEle-1][3])
    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)

    K = np.array( diags([diagB,diagA,diagA], [0,-1, 1]).todense() )
    return(K,F)
# function to find k and f for Quadratic elements
def kfmatixQad(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    xc=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    for i in range(nEle):
        xc.append(0.5*xa[i]+0.5*xb[i])
    # print('xa',xa)
    # print('xb',xb)
    # print('xc',xc)
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = ((x-xc[i])*(x-xb[i]))/((xa[i]-xc[i])*(xa[i]-xb[i]))
        psi_2 = ((x-xa[i])*(x-xb[i]))/((xc[i]-xa[i])*(xc[i]-xb[i]))
        psi_3 = ((x-xa[i])*(x-xc[i]))/((xb[i]-xa[i])*(xb[i]-xc[i]))
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_3 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_3,x)+c*psi_1*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_3,x)+c*psi_2*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_1,x)+c*psi_3*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_2,x)+c*psi_3*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_3,x)+c*psi_3*psi_3,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    F.append(Ftemp[1])
    t=0
```

```
    i=0
    while t<nEle-1:
        F.append(Ftemp[i+2]+Ftemp[i+3])
        F.append(Ftemp[i+4])
        i=i+3
        t=t+1
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # print('len f',len(F))
    # three diagonals of the quadiagonal matrix
    diagA=[]
    diagB=[]
    diagC=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
        diagA.append(k[i][5])
    # print('diagA',diagA)

    for i in range(nEle):
        diagC.append(k[i][2])
        if i!=nEle-1:
            diagC.append(0.0)
    # print('diagC',diagC)

    diagB.append(k[0][0])
    diagB.append(k[0][4])
    for i in range(nEle-1):
        diagB.append(k[i][8]+k[i+1][0])
        diagB.append(k[i+1][4])
    diagB.append(k[nEle-1][8])

    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)
    diagC=np.array(diagC, dtype=np.float64)

    K = np.array(diags([diagB,diagA,diagA,diagC,diagC], [0,-1, 1,-2, 2]).todense
() )
    return(K,F)
# function to find Q at the left and right ends
def solQ(k,f,nEle,bcs,method):
    bc1=Symbol('bc1')
    bc2=Symbol('bc2')
    q1=Symbol('q1')
    q2=Symbol('q2')
    c1=0
    c2=0
    if method=='linear':
        for i in range(nEle+1):
            c1=c1+k[0][i]
            c2=c2+k[nEle][i]
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[nEle]-q2
    elif method=='Quadratic':
        for i in range(2*nEle+1):
            c1=c1+k[0][i]
            c2=c2+k[nEle][i]
```

```
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[2*nEle]-q2
    if len(bcs)==0:
        return(solve(q1_eqn,q1),solve(q2_eqn,q2))
    elif len(bcs)==2:
        return(solve(q1_eqn,q1)[0].subs(bc1,bcs[0]),solve(q2_eqn,q2)[0].subs(bc2
,bcs[1]))

# user inputs
domainLen=1
# number of elements
nEle=3
# problem data
a=1
c=-1
f=-x*x
method='Both' # accepts 'Quadratic','linear','Both'
# dirchlet boundary conditions values at left and right end
bcs=[2,3] # assumed boundary values [left end,right end]

if method=='Quadratic':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)

elif method=='linear':
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('Linear Solution, U : \n',sl)

elif method=='Both':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('\nLinear Solution, U : \n',sl)
```

```
Quadratic Solution, U :
 [1.03201462003385 0.999225763913502 0.939680884785393 0.85726861573
8989
 0.758188394886948 0.650514484500336 0.544211364016817]

Linear Solution, U :
 [0.601031384163611 0.461177147047616 0.285154089849659 0.1285283642
64063]
```

In [14]:

```
# Part (b) BVP 1
# Hear transfer through fin

from sympy.solvers import solve
from sympy import Symbol
from sympy import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import *
from numpy.linalg import inv
from array import *
from scipy import linalg
x=Symbol('x')

# function to find k and f for linear elements
def kfmatixLin(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = (xb[i]-x)/(xb[i]-xa[i])
        psi_2 = (x-xa[i])/(xb[i]-xa[i])
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    for i in range(0,len(Ftemp)-2,2):
        F.append(Ftemp[i+1]+Ftemp[i+2])
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # two diagonals of the tridiagonal matrix
    diagA=[]
    diagB=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
    # print('diagA',diagA)
    # NOTE: no need for diagC as it will always be same as diagA
```

```
    diagB.append(k[0][0])
    for i in range(nEle-1):
        diagB.append(k[i][3]+k[i+1][0])
    diagB.append(k[nEle-1][3])
    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)

    K = np.array( diags([diagB,diagA,diagA], [0,-1, 1]).todense() )
    return(K,F)
# function to find k and f for Quadratic elements
def kfmatixQad(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    xc=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    for i in range(nEle):
        xc.append(0.5*xa[i]+0.5*xb[i])
    # print('xa',xa)
    # print('xb',xb)
    # print('xc',xc)
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = ((x-xc[i])*(x-xb[i]))/((xa[i]-xc[i])*(xa[i]-xb[i]))
        psi_2 = ((x-xa[i])*(x-xb[i]))/((xc[i]-xa[i])*(xc[i]-xb[i]))
        psi_3 = ((x-xa[i])*(x-xc[i]))/((xb[i]-xa[i])*(xb[i]-xc[i]))
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_3 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_3,x)+c*psi_1*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_3,x)+c*psi_2*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_1,x)+c*psi_3*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_2,x)+c*psi_3*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_3,x)+c*psi_3*psi_3,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    F.append(Ftemp[1])
    t=0
```

```
    i=0
    while t<nEle-1:
        F.append(Ftemp[i+2]+Ftemp[i+3])
        F.append(Ftemp[i+4])
        i=i+3
        t=t+1
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # print('len f',len(F))
    # three diagonals of the quadiagonal matrix
    diagA=[]
    diagB=[]
    diagC=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
        diagA.append(k[i][5])
    # print('diagA',diagA)

    for i in range(nEle):
        diagC.append(k[i][2])
        if i!=nEle-1:
            diagC.append(0.0)
    # print('diagC',diagC)

    diagB.append(k[0][0])
    diagB.append(k[0][4])
    for i in range(nEle-1):
        diagB.append(k[i][8]+k[i+1][0])
        diagB.append(k[i+1][4])
    diagB.append(k[nEle-1][8])

    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)
    diagC=np.array(diagC, dtype=np.float64)

    K = np.array(diags([diagB,diagA,diagA,diagC,diagC], [0,-1, 1,-2, 2]).todense
() )
    return(K,F)
# function to find Q at the left and right ends
def solQ(k,f,nEle,bcs,method):
    bc1=Symbol('bc1')
    bc2=Symbol('bc2')
    q1=Symbol('q1')
    q2=Symbol('q2')
    c1=0
    c2=0
    if method=='linear':
        for i in range(nEle+1):
            c1=c1+k[0][i]
            c2=c2+k[nEle][i]
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[nEle]-q2
    elif method=='Quadratic':
        for i in range(2*nEle+1):
            c1=c1+k[0][i]
            c2=c2+k[nEle][i]
```

```
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[2*nEle]-q2
    if len(bcs)==0:
        return(solve(q1_eqn,q1),solve(q2_eqn,q2))
    elif len(bcs)==2:
        return(solve(q1_eqn,q1)[0].subs(bc1,bcs[0]),solve(q2_eqn,q2)[0].subs(bc2
,bcs[1]))

# user inputs__________________________________________________________________
domainLen=1
# number of elements
nEle=10
# problem data
a=0.1  # a= kA
c=0.021  # c = P*beta
f=2.5    # f = P*beta*T_infinity
method='Both' # accepts 'Quadratic','linear','Both'
# dirchlet boundary conditions values at left and right end
bcs=[125,80] # assumed boundary values [left end,right end]
# user inputs__________________________________________________________________


if method=='Quadratic':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)

elif method=='linear':
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('Linear Solution, U : \n',sl)

elif method=='Both':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('\nLinear Solution, U : \n',sl)
```

```
Quadratic Solution, U :
 [119.600786870395 119.601973839834 119.603451860820 119.60522170424
0
 119.607284304424 119.609640739131 119.612292250708 119.615240226036
 119.618486218102 119.622031925884 119.625879216215 119.630030103669
 119.634486772945 119.639251558629 119.644326967572 119.649715659239
 119.655420468636 119.661444385969 119.667790578854 119.674462372815
 119.681463276846]

Linear Solution, U :
 [116.844160630367 116.848098377241 116.847415513492 116.84211060460
7
 116.832172506375 116.817580341483 116.798303455655 116.774301353259
 116.745523612231 116.711909778156 116.673389237267]
```

In [17]:

```
# Part (b) BVP 2
# 1-D Heat transfer


from sympy.solvers import solve
from sympy import Symbol
from sympy import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import *
from numpy.linalg import inv
from array import *
from scipy import linalg
x=Symbol('x')

# function to find k and f for linear elements
def kfmatixLin(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = (xb[i]-x)/(xb[i]-xa[i])
        psi_2 = (x-xa[i])/(xb[i]-xa[i])
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    for i in range(0,len(Ftemp)-2,2):
        F.append(Ftemp[i+1]+Ftemp[i+2])
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # two diagonals of the tridiagonal matrix
    diagA=[]
    diagB=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
    # print('diagA',diagA)
    # NOTE: no need for diagC as it will always be same as diagA
```

```
    diagB.append(k[0][0])
    for i in range(nEle-1):
        diagB.append(k[i][3]+k[i+1][0])
    diagB.append(k[nEle-1][3])
    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)

    K = np.array( diags([diagB,diagA,diagA], [0,-1, 1]).todense() )
    return(K,F)
# function to find k and f for Quadratic elements
def kfmatixQad(nEle,domainLen,a,c,f):
    xb=[]
    xa=[]
    xc=[]
    for i in range(1,nEle+1):
        xb.append(i*(domainLen/nEle))
        xa.append((i-1)*(domainLen/nEle))
    for i in range(nEle):
        xc.append(0.5*xa[i]+0.5*xb[i])
    # print('xa',xa)
    # print('xb',xb)
    # print('xc',xc)
    k=[]
    Ftemp=[]
    # for ith element  # NOTE: [[element 1 k's],[element 2 k's], ...]
    for i in range(nEle):
        psi_1 = ((x-xc[i])*(x-xb[i]))/((xa[i]-xc[i])*(xa[i]-xb[i]))
        psi_2 = ((x-xa[i])*(x-xb[i]))/((xc[i]-xa[i])*(xc[i]-xb[i]))
        psi_3 = ((x-xa[i])*(x-xc[i]))/((xb[i]-xa[i])*(xb[i]-xc[i]))
        # print(psi_1)
        # print(psi_2)
        k.append([])
        Ftemp.append(integrate(f*psi_1 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_2 ,(x, xa[i], xb[i])))
        Ftemp.append(integrate(f*psi_3 ,(x, xa[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_1,x)+c*psi_1*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_2,x)+c*psi_1*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_1,x)*diff(psi_3,x)+c*psi_1*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_1,x)+c*psi_2*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_2,x)+c*psi_2*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_2,x)*diff(psi_3,x)+c*psi_2*psi_3,(x, x
a[i], xb[i])))

        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_1,x)+c*psi_3*psi_1,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_2,x)+c*psi_3*psi_2,(x, x
a[i], xb[i])))
        k[i].append(integrate( a*diff(psi_3,x)*diff(psi_3,x)+c*psi_3*psi_3,(x, x
a[i], xb[i])))

    F=[]
    F.append(Ftemp[0])
    F.append(Ftemp[1])
```

```
    t=0
    i=0
    while t<nEle-1:
        F.append(Ftemp[i+2]+Ftemp[i+3])
        F.append(Ftemp[i+4])
        i=i+3
        t=t+1
    F.append(Ftemp[len(Ftemp)-1])

    # print('k = ',k)
    # print('Ftemp',Ftemp)
    # print('F = ',F)
    # print('len f',len(F))
    # three diagonals of the quadiagonal matrix
    diagA=[]
    diagB=[]
    diagC=[]
    # for three element 4*4 k matrix
    for i in range(nEle):
        diagA.append(k[i][1])
        diagA.append(k[i][5])
    # print('diagA',diagA)

    for i in range(nEle):
        diagC.append(k[i][2])
        if i!=nEle-1:
            diagC.append(0.0)
    # print('diagC',diagC)

    diagB.append(k[0][0])
    diagB.append(k[0][4])
    for i in range(nEle-1):
        diagB.append(k[i][8]+k[i+1][0])
        diagB.append(k[i+1][4])
    diagB.append(k[nEle-1][8])

    # print('diagB',diagB)
    diagA=np.array(diagA, dtype=np.float64)
    diagB=np.array(diagB, dtype=np.float64)
    diagC=np.array(diagC, dtype=np.float64)

    K = np.array(diags([diagB,diagA,diagA,diagC,diagC], [0,-1, 1,-2, 2]).todense
() )
    return(K,F)
# function to find Q at the left and right ends
def solQ(k,f,nEle,bcs,method):
    bc1=Symbol('bc1')
    bc2=Symbol('bc2')
    q1=Symbol('q1')
    q2=Symbol('q2')
    c1=0
    c2=0
    if method=='linear':
        for i in range(nEle+1):
            c1=c1+k[0][i]
            c2=c2+k[nEle][i]
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[nEle]-q2
    elif method=='Quadratic':
        for i in range(2*nEle+1):
            c1=c1+k[0][i]
```

```
            c2=c2+k[nEle][i]
        q1_eqn = c1*bc1-f[0]+q1
        q2_eqn = c2*bc2-f[2*nEle]-q2
    if len(bcs)==0:
        return(solve(q1_eqn,q1),solve(q2_eqn,q2))
    elif len(bcs)==2:
        return(solve(q1_eqn,q1)[0].subs(bc1,bcs[0]),solve(q2_eqn,q2)[0].subs(bc2
,bcs[1]))

# user inputs_____________________________________________________________________
domainLen=1
# number of elements
nEle=10
# problem data
a=0.1  # a= kA
c=0.02  # c = Aq+P*beta
f=3.8    # f = P*beta*T_infinity
method='Both' # accepts 'Quadratic','linear','Both'
# dirchlet boundary conditions values at left and right end
bcs=[195,180] # assumed boundary values [left end,right end]
# user inputs_____________________________________________________________________


if method=='Quadratic':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)

elif method=='linear':
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('Linear Solution, U : \n',sl)

elif method=='Both':
    kq,fq=kfmatixQad(nEle,domainLen,a,c,f)
    q1q,q2q=solQ(kq,fq,nEle,bcs,'Quadratic')
    fq[0]=fq[0]+q1q
    fq[len(fq)-1]=fq[len(fq)-1]+q2q
    sq=linalg.inv(kq).dot(fq)
    print('Quadratic Solution, U : \n',sq)
    k,f=kfmatixLin(nEle,domainLen,a,c,f)
    q1,q2 = solQ(k,f,nEle,bcs,'linear')
    f[0]=f[0]+q1
    f[nEle]=f[nEle]+q2
    sl=linalg.inv(k).dot(f)
    print('\nLinear Solution, U : \n',sl)
```

```
Quadratic Solution, U :
 [192.652230902594 192.653727385186 192.656550797817 192.66070253011
0
 192.666184680188 192.672999967030 192.681151820677 192.690644294893
 192.701482158532 192.713670808274 192.727216361268 192.742125567968
 192.758405906259 192.776065493484 192.795113183078 192.815558475906
 192.837411617594 192.860683512384 192.885385820997 192.911530870612
 192.939131757948]

Linear Solution, U :
 [189.249962189202 189.254213568518 189.256972877613 189.25824563694
8
 189.258034392889 189.256338722806 189.253155234230 189.248477558059
 189.242296335824 189.234599200956 189.225370754053]
```

In [ ]: