MA347: Numerical Solution of PDE's

Assignment 2

a)

$u''(x) - 10u(x) = \frac{1}{6}[e^x - 1], \quad 0 \leq x \leq 3, u(3) = 4, \quad u'(0) = 8$

Multiplying both sides through using an arbitrary weight function, w(x)

$w(x)u''(x) - w(x)10u(x) = w(x)\frac{1}{6}[e^x - 1]$

Integrating over the $0 \leq x \leq 3$ domain and reducing to the weak form of the problem.

$$\int_0^3 w(x)\, u''(x)dx - \int_0^3 w(x)\, 10u(x)dx = \int_0^3 w(x)\frac{1}{6}[e^x - 1]dx$$

Where $\int_0^3 w(x)\, u''(x)dx = [u'(x)w(x)]_{x=0}^3 - \int_0^3 w'(x)\, u'(x)dx$

= $u'(3)w(3) - u'(0)w(0) - \int_0^3 w'(x)\, u'(x)dx$

=$4w(0) - \int_0^3 w'(x)\, u'(x)dx$

Substituting back into the original equation yields:

$$4w(0) - \int_0^3 w'(x)\, u'(x)dx - \int_0^3 w(x)\, 10u(x)dx = \int_0^3 w(x)\frac{1}{6}[e^x - 1]dx$$

Rearranging:

$$-\int_0^3 w'(x)\, u'(x)dx - \int_0^3 w(x)\, 10u(x)dx = \int_0^3 w(x)\frac{1}{6}[e^x - 1]dx - 4w(0)$$

Which can be transformed into:

$\big(u'(x), w'(x)\big) + 10(w(x), u(x) = -(\frac{1}{6}[e^x - 1], w(x)) - 4w(0))$

Now, Approximating u(x) and w(x) as the sum of ϕ shape functions:

$$w(x) = \sum_{i=0}^{M-1} w_i\, \Phi_i(x), \qquad u(x) = \sum_{j=0}^{M-1} uj\, \Phi_j(x) + q\Phi_m(x)$$

$$\sum_{i=0}^{M-1} w_i{}'\, \Phi_i{}'(x) \sum_{j=0}^{M-1} u_j\, \Phi'_j(x) + 4\Phi'_m(x) - \ 10 \sum_{i=0}^{M-1} w_i\, \Phi_i(x) \sum_{j=0}^{M-1} uj\, \Phi_j(x) + 4\Phi_m(x)$$

$$= \left(\left(\frac{1}{6}[e^x - 1]\right), \left(\sum_{i=0}^{M-1} w_i\, \Phi_i(x) \right) \right) - 4 \sum_{i=0}^{M-1} w_i\, \Phi_i(0)$$

Simplifying yields:

$$\sum_{i=0}^{M-1}\sum_{j=0}^{M-1} w_i u_j(\Phi'_i\Phi'_j) + 4w_i(\Phi'_i\Phi'_m) - 10w_i u_j(\Phi_i\Phi_m) + 4w_i(\Phi_i\Phi_m) = \sum_{i=0}^{M-1} w_i(\frac{1}{6}[e^x - 1], \Phi_i) - 4w_i\Phi_i(0)$$

$$= \sum_{i=0}^{M-1} w_i \left[\sum_{j=0}^{M-1} u_j(\Phi'_i\Phi'_j) + 4(\Phi'_i\Phi'_m) - 10u_j(\Phi_i\Phi_j) + 4(\Phi_i\Phi_m) - \left(\frac{1}{6}[e^x - 1], \Phi_i\right) + 4\Phi_i(0)\right] = 0$$

As each w_i is arbitrary we can further reduce to:

$$\sum_{j=0}^{M-1} u_j[(\Phi'_i\Phi'_j) - 10(\Phi_i\Phi_j)] = -4(\Phi_i\Phi_m) - 4(\Phi'_i\Phi'_m) + \left(\frac{1}{6}[e^x - 1], \Phi_i\right) - 4\Phi_i(0)$$

Where the force vector is given by: $F_i = -4(\Phi_i\Phi_m) - 4(\Phi'_i\Phi'_m) + \left(\frac{1}{6}[e^x - 1], \Phi_i\right) - 4\Phi_i(0)$

and the stiffness matrix is given by: $K = (\Phi'_i\Phi'_j) - 10(\Phi_i\Phi_j), \quad i, j = 0,1, \dots M - 1$

b)

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.linalg as la #Linear algebra solver
import time #Measure the time

time_start = time.time() #Begin counter to record the computation time

df = 10 #Degrees of freedom
c = (-1/6) #Value of constant in the ODE
h = 8 #Value of derivative at x=0
q = 4 #Value at boundary x=1
Xmin = 0 #Minimum value of x
Xmax = 3 #Maximum value of x

xi = np.linspace(Xmin, Xmax, df+1) #Define x_m, which defines the shape functions phi_j

N = 5000 #Number of nodes for 'continuous' x. Set arbitrarily high to ensure accuracy.
x = np.linspace(Xmin, Xmax, N) #Define x
hh = x[3] - x[0] #The difference between the points in continuous x, needed for the trapezium rule.


def RHS(x,c):

    return c*((np.exp(x))-1)

from shape_functions_2020_03_14 import phi_j #The shape function phi_j
from shape_functions_2020_03_14 import dphi_j #The derivative of the shape function phi'_j
from shape_functions_2020_03_14 import int_trap #The integral function (f,g), using the trapezium rule

intPhiM = np.zeros(np.linspace(0,1,df).shape) #Integral term (phi'_M,phi'_j)
for j in range(0,df):
    intPhiM[j] = int_trap(x,phi_j(x,xi,df,df),phi_j(x,xi,j,df),hh)

intdPhiM = np.zeros(np.linspace(0,1,df).shape) #Integral term (phi'_M,phi'_j)
for j in range(0,df):
    intdPhiM[j] = int_trap(x,dphi_j(x,xi,df,df),dphi_j(x,xi,j,df),hh)

intPhif = np.zeros(np.linspace(0,1,df).shape) #Integral term (phi_j,f)
for j in range(0,df):
    intPhif[j] = int_trap(x,phi_j(x,xi,j,df),RHS(x,c),hh)


phij0 = np.zeros(np.linspace(0,1,df).shape) #Term phi_j (0)
phij0[0] = xi[1]/(xi[1]-xi[0]) #We only need to compute for phi_0 as all others are zero

ForceV = (-4)*(intPhiM - intdPhiM - phij0 ) - intPhif

StiffnessM = np.zeros(np.linspace(0,1,df).shape) #The stiffness matrix
int_itt1 = np.zeros(np.linspace(0,1,df).shape) #An iteration parameter
int_itt2 = np.zeros(np.linspace(0,1,df).shape)
int_itt3 = np.zeros(np.linspace(0,1,df).shape)

for j in range(0,df): #Produces the first row of the stiffness matrix K_0j
    int_itt1[j] = int_trap(x,dphi_j(x,xi,0,df),dphi_j(x,xi,j,df),hh)
    int_itt2[j] = int_trap(x,phi_j(x,xi,0,df),phi_j(x,xi,j,df),hh)
    StiffnessM[j] = int_itt1[j] - 10* int_itt2[j]
```

```
58      int_itt1 = np.zeros(np.linspace(0,1,df).shape)
59      int_itt2 = np.zeros(np.linspace(0,1,df).shape)
60      #Produces remaining rows, then adds them to the bottom of the stiffness matrix
61      for i in range(1,df):
62          for j in range(0,df): #Produces the next row K_ij for fixed i
63              int_itt1[j] = int_trap(x,dphi_j(x,xi,i,df),dphi_j(x,xi,j,df),hh)
64              int_itt2[j] = int_trap(x,phi_j(x,xi,i,df),phi_j(x,xi,j,df),hh)
65              int_itt3[j] = int_itt1[j] - 10* int_itt2[j]
66
67          StiffnessM = np.vstack([StiffnessM,int_itt3]) #Adds the new row to the bottom of the matrix
68
69      from shape_functions_2020_03_14 import Banded_solve
70
71      ui = Banded_solve(StiffnessM,ForceV,df)   #Solve the linear algebra problem: K u = F noting that K is banded
72
73      ux = 0 #Create: u(x), our numerical solution
74
75      for j in range(0,df+1):
76          if j == df: #Add the term q \phi_M(x)
77              ux = ux + phi_j(x,xi,j,df)
78          else: #Add all remaining terms u_j \phi_j(x), j=1,...,M-1
79              ux = ux + ui[j]*phi_j(x,xi,j,df)
80
81      time_elapsed = (time.time() - time_start) #Determine the total computation time
82
83      print("degrees of freedom =", df) #Print the degrees of freedom
84      print("computation time=", '%.3f' % time_elapsed, "seconds") #Print the computation time (to 2d.p.)
85
86      fig1 = plt.figure(figsize=(5,5.5))
87
88      plt.plot(x, ux,'r--',label = "FE approximation")
89      plt.xlabel('x') #Label the x-axis
90      plt.ylabel('u(x)') #Label the y-axis
91      plt.title('degrees of freedom: %i' %df) #Add a title, stating the degrees of freedom
92      plt.legend(loc="upper right")  #Add a legend in the top right corner
93      #fig1.savefig('FE_ProbSh4_apx4.pdf', format = 'pdf')
94      plt.show() #Show the current figure, begin the next figure
95
96
```

The code begins by importing relevant/necessary libraries, defining our variables based on the given PDE and then defining the phi_j shape functions and x.

Line 22 to 24 defines the right hand side of the given PDE

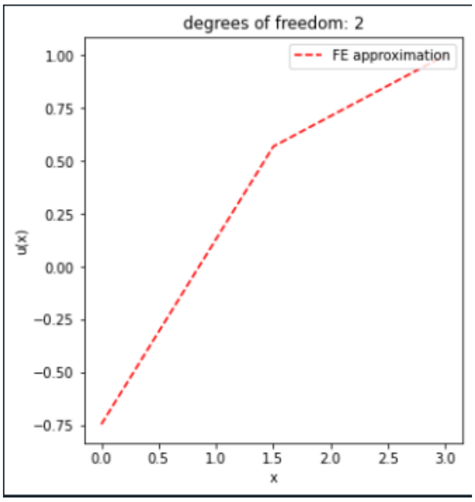
26 – 46 is the importation of the shape functions from the given code, as well as the calculation of each integral term in order to produce the force vector F

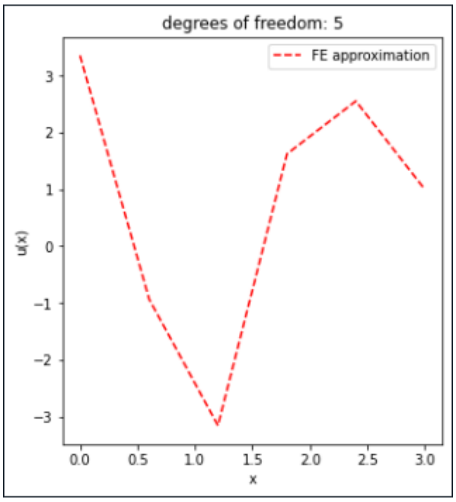
48 – 67 is where the stiffness matrix K is created by producing rows and stacking them.

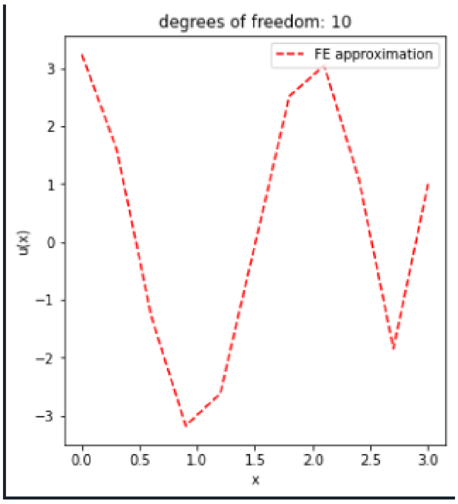
69 – 79 is solving the Ku = F problem using the given shape functions code, and then creating our numerical solution from combining the above code.

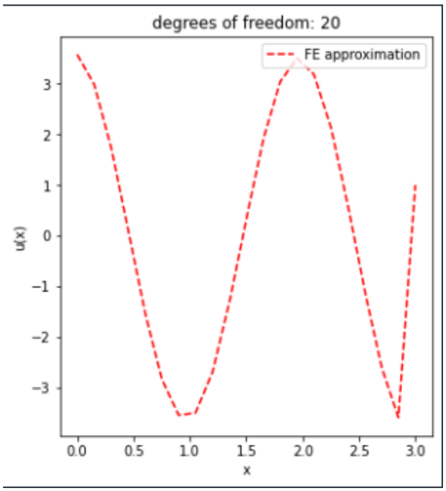
80-94 is simply graphing the results and printing the degrees of freedom with the computation time.

c)






degrees of freedom: 10
FE approximation
u(x)
x


degrees of freedom: 20
FE approximation
u(x)
x