﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Mar 9 11:24:56 2022

@author: juanmeriles

"""

import numpy as np

import matplotlib.pyplot as plt

import scipy.optimize as sc

import copy

nodes = np.array ([[0,0],

[0,4],

[0,8],

[0,12],

[4,0],

[5,5],

[3,9],

[4,12],

[8,0],

[7,4],

[9,7],

[8,12],

[12,0],

[13,3],

[11,8],

[12,12],

[16,0],

[16,4],

[16,8],

[16,12]])

CON = np.array([[1,2,6,5],

[2,3,7,6],

[3,4,8,7],

[5,6,10,9],

[6,7,11,10],

[7,8,12,11],

[9,10,14,13],

[10,11,15,14],

[11,12,16,15],

[13,14,18,17],

[14,15,19,18],

[15,16,20,19]])

BOUN = np.array([1,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,1,1])

def MeanRemap(nodesMean,CON,BOUN):

newnodes = []

for i in range(len(nodesMean)):

surel = []

if BOUN[i] == 1:

newnodes.append(nodesMean[i])

else:

for j in range(len(CON)):

for k in range(len(CON[0])):

if (CON[j][k] == i+1):

surel.append(np.delete(CON[j],k))

surnodes = np.zeros(8)

count = 0

for j in range(len(surel)):

for k in range(len(surel[0])):

if (surel[j][k] not in surnodes):

surnodes[count] = surel[j][k]

count = count+1

newx = 0

newy = 0

for j in range(len(surnodes)):

newx = newx+nodesMean[int(surnodes[j])-1][0]

newy = newy+nodesMean[int(surnodes[j])-1][1]

newx = newx/len(surnodes)

newy = newy/len(surnodes)

#nodesMean[i][0] = newx

#nodesMean[i][1] = newy

newnodes.append(np.array([newx,newy]))

newnodes = np.vstack(newnodes)

return newnodes

def PoisEqs(node,x,y):

xe = 1/2\*(x[7]-x[3])

xn = 1/2\*(x[1]-x[5])

xee = x[7]-2\*node[0]+x[3]

xnn = x[1]-2\*node[0]+x[5]

xen = (1/4)\*(x[0]+x[4]-x[6]-x[2])

ye = 1/2\*(y[7]-y[3])

yn = 1/2\*(y[1]-y[5])

yee = y[7]-2\*node[1]+y[3]

ynn = y[1]-2\*node[1]+y[5]

yen = (1/4)\*(y[0]+y[4]-y[6]-y[2])

alpha = xn\*\*2+yn\*\*2

beta = xe\*xn+ye\*yn

gamma = xe\*\*2+ye\*\*2

f1 = alpha\*xee-2\*beta\*xen+gamma\*xnn

f2 = alpha\*yee-2\*beta\*yen+gamma\*ynn

return [f1,f2]

def PoissonRemap(nodesPois,CON,BOUN):

newnodes = []

nodesPois = nodesPois.astype('float')

for i in range(len(nodesPois)):

surel = []

if BOUN[i] == 1:

newnodes.append(nodesPois[i])

else:

surnodes = np.zeros(8)

for j in range(len(CON)):

for k in range(len(CON[0])):

if (CON[j][k] == i+1):

surel.append(CON[j])

if k==0:

surnodes[0] = CON[j][2]

surnodes[1] = CON[j][1]

surnodes[7] = CON[j][3]

if k==1:

surnodes[5] = CON[j][0]

surnodes[6] = CON[j][3]

if k==2:

surnodes[3] = CON[j][1]

surnodes[4] = CON[j][0]

if k==3:

surnodes[2] = CON[j][1]

node = [nodesPois[i][0],nodesPois[i][1]]

surnodes = surnodes - np.ones(len(surnodes))

#print(surnodes)

x = nodesPois[surnodes.astype(int)].T[0]

y = nodesPois[surnodes.astype(int)].T[1]

[xnew,ynew] = sc.fsolve(PoisEqs,node,(x,y))

nodesPois[i][0] = xnew

nodesPois[i][1] = ynew

newnodes.append(np.array([xnew,ynew]))

newnodes = np.vstack(newnodes)

return newnodes

def plotElements(inputNodes,CON,color):

for j in range(len(CON)):

y = []

x = []

for i in range(len(CON[0])):

x.append(inputNodes[CON[j][i]-1][0])

y.append(inputNodes[CON[j][i]-1][1])

x.append(x[0])

y.append(y[0])

plt.plot(x,y,color)

plotElements(nodes,CON,'b')

meanNodes = copy.copy(nodes)

#print(nodes)

poisNodes = copy.copy(nodes)

nodestrue = np.array ([[0,0],

[0,4],

[0,8],

[0,12],

[4,0],

[4,4],

[4,8],

[4,12],

[8,0],

[8,4],

[8,8],

[8,12],

[12,0],

[12,4],

[12,8],

[12,12],

[16,0],

[16,4],

[16,8],

[16,12]])

errorMean = []

errorPois = []

for i in range(25):

meanNodes = MeanRemap(meanNodes,CON,BOUN)

poisNodes = PoissonRemap(poisNodes,CON,BOUN)

errorMean.append(np.linalg.norm(meanNodes-nodestrue,2))

errorPois.append(np.linalg.norm(poisNodes-nodestrue,2))

plotElements(meanNodes,CON,'r')

plotElements(poisNodes,CON,'g')

plt.figure(2)

plt.plot(errorMean,label = 'Neighbor Average')

plt.plot(errorPois, label = 'Poisson Smoothing')

plt.legend()

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Wed Mar 9 11:24:56 2022

@author: juanmeriles

"""

import numpy as np

import matplotlib.pyplot as plt

import scipy.optimize as sc

import copy

nodesnat = np.array ([[0,0],

[0,4],

[0,8],

[4,0],

[4,4],

[4,8],

[8,0],

[8,4],

[8,8],

[12,0],

[12,4],

[12,8]])

nodes = np.array ([[0,0],

[0,4],

[0,8],

[4,0],

[5,3],

[4,8],

[8,0],

[9,5],

[8,8],

[12,0],

[12,4],

[12,8]])

con = np.array([[1,4,5,2],

[2,5,6,3],

[4,7,8,5],

[5,8,9,6],

[7,10,11,8],

[8,11,12,9]])

boun = np.array([1,1,1,1,0,1,1,0,1,1,1,1])

class element:

NODE = []

CON = []

BOUN = []

id\_v = []

def \_\_init\_\_(self):

pass

def ShapeFcn(e,n):

N1 = 1/4\*(1-e)\*(1-n)

N2 = 1/4\*(1+e)\*(1-n)

N3 = 1/4\*(1+e)\*(1+n)

N4 = 1/4\*(1-e)\*(1+n)

N = np.array([[N1,0,N2,0,N3,0,N4,0],

[0,N1,0,N2,0,N3,0,N4]])

Nsmall = np.array([[N1,N2,N3,N4]])

return N, Nsmall

def Jacobian(e,n,el):

dxde = 1/4\*(-(1-n)\*el.NODE[0][0]+(1-n)\*el.NODE[1][0]+(1+n)\*el.NODE[2][0]-(1+n)\*el.NODE[3][0])

dyde = 1/4\*(-(1-n)\*el.NODE[0][1]+(1-n)\*el.NODE[1][1]+(1+n)\*el.NODE[2][1]-(1+n)\*el.NODE[3][1])

dxdn = 1/4\*(-(1-e)\*el.NODE[0][0]-(1+e)\*el.NODE[1][0]+(1+e)\*el.NODE[2][0]+(1-e)\*el.NODE[3][0])

dydn = 1/4\*(-(1-e)\*el.NODE[0][1]-(1+e)\*el.NODE[1][1]+(1+e)\*el.NODE[2][1]+(1-e)\*el.NODE[3][1])

J = np.array([[dxde,dyde],

[dxdn,dydn]])

return J

def plotElements(inputNodes,CON,color):

for j in range(len(CON)):

y = []

x = []

for i in range(len(CON[0])):

x.append(inputNodes[CON[j][i]-1][0])

y.append(inputNodes[CON[j][i]-1][1])

x.append(x[0])

y.append(y[0])

plt.plot(x,y,color)

def func(nap,x,el):

N1 = 1/4\*(1-nap[0])\*(1-nap[1])

N2 = 1/4\*(1+nap[0])\*(1-nap[1])

N3 = 1/4\*(1+nap[0])\*(1+nap[1])

N4 = 1/4\*(1-nap[0])\*(1+nap[1])

N = np.array([[N1,0,N2,0,N3,0,N4,0],

[0,N1,0,N2,0,N3,0,N4]])

return (N@el.nodeVec.T).flatten()-x

def whichel(x,y):

if x<=4 and y<=4:

inel = 1

elif x<=4 and y<=8:

inel = 2

elif x<=8 and y<=4:

inel = 3

elif x<=8 and y<=8:

inel = 4

elif x<=12 and y<=4:

inel = 5

elif x<=12 and y<=8:

inel = 6

else:

inel = -1

print('error')

return inel

#def GaussInt():

el\_og = []

el\_trans = []

numel = len(con)

for i in range(numel):

el\_og.insert(i,element())

el\_og[i].NODE = np.array([nodesnat[con[i][0]-1],nodesnat[con[i][1]-1],nodesnat[con[i][2]-1],nodesnat[con[i][3]-1]])

el\_og[i].CON = con[i]

el\_og[i].BOUN = np.array([boun[con[i][0]-1],boun[con[i][1]-1],boun[con[i][2]-1],boun[con[i][3]-1]])

el\_og[i].nodeVec = np.array([[el\_og[i].NODE[0][0],el\_og[i].NODE[0][1],el\_og[i].NODE[1][0],el\_og[i].NODE[1][1],\

el\_og[i].NODE[2][0],el\_og[i].NODE[2][1],el\_og[i].NODE[3][0],el\_og[i].NODE[3][1]]])

for i in range(numel):

el\_trans.insert(i,element())

el\_trans[i].NODE = np.array([nodes[con[i][0]-1],nodes[con[i][1]-1],nodes[con[i][2]-1],nodes[con[i][3]-1]])

el\_trans[i].CON = con[i]

el\_trans[i].BOUN = np.array([boun[con[i][0]-1],boun[con[i][1]-1],boun[con[i][2]-1],boun[con[i][3]-1]])

za = np.array([1,1,1,1,1,1,1,1,1,1,1,1])

zb = np.array([0,8,16,4,12,20,8,16,24,12,20,28])

zc = np.array([2,8,-7,6,-1,1,3,-2,4,-4,-3,0])

globalGaussPointsx\_og = []

globalGaussPointsy\_og = []

globalGaussPointsx\_trans = []

globalGaussPointsy\_trans = []

A = np.zeros((12,12))

b1 = np.zeros((12,1))

b2 = np.zeros((12,1))

b3 = np.zeros((12,1))

count = 0

for i in range(numel):

gaussPoints = [-np.sqrt(3/5),0,np.sqrt(3/5)]

w = [5/9,8/9,5/9]

for j in range(len(w)):

for k in range(len(w)):

nodeVec\_og = np.array([[el\_og[i].NODE[0][0],el\_og[i].NODE[0][1],el\_og[i].NODE[1][0],el\_og[i].NODE[1][1],\

el\_og[i].NODE[2][0],el\_og[i].NODE[2][1],el\_og[i].NODE[3][0],el\_og[i].NODE[3][1]]])

nodeVec\_trans = np.array([[el\_trans[i].NODE[0][0],el\_trans[i].NODE[0][1],el\_trans[i].NODE[1][0],el\_trans[i].NODE[1][1],\

el\_trans[i].NODE[2][0],el\_trans[i].NODE[2][1],el\_trans[i].NODE[3][0],el\_trans[i].NODE[3][1]]])

N\_og, Nsmall\_og = ShapeFcn(gaussPoints[j],gaussPoints[k])

N\_trans, Nsmall\_trans = ShapeFcn(gaussPoints[j],gaussPoints[k])

globalGaussPointsx\_og.append((N\_og @ nodeVec\_og.T)[0])

globalGaussPointsy\_og.append((N\_og @ nodeVec\_og.T)[1])

globalGaussPointsx\_trans.append((N\_trans @ nodeVec\_trans.T)[0])

globalGaussPointsy\_trans.append((N\_trans @ nodeVec\_trans.T)[1])

J\_trans = np.linalg.det(Jacobian(gaussPoints[j],gaussPoints[k],el\_trans[i]))

gp\_global = [globalGaussPointsx\_trans[count][0],globalGaussPointsy\_trans[count][0]]

zel1 = 1

zel2 = globalGaussPointsx\_trans[count]+2\*globalGaussPointsy\_trans[count]

#which element are we in

inel = whichel(globalGaussPointsx\_trans[count],globalGaussPointsy\_trans[count])

#print(inel)

#find natural coords of gp

guess = np.array([0,0])

natcoords = sc.fsolve(func,guess,(gp\_global,el\_og[inel-1]))

#print(gp\_global)

#get interpolated values

ztemp = np.array([[zc[con[inel-1][0]-1],zc[con[inel-1][1]-1],zc[con[inel-1][2]-1],zc[con[inel-1][3]-1]]])

#print(inel)

#print(natcoords)

#print(ztemp)

SFtemp,SFtempsmall = ShapeFcn(natcoords[0],natcoords[1])

zel3 = SFtempsmall @ ztemp.T

count = count+1

Atemp = Nsmall\_trans.T@ Nsmall\_trans

btemp1 = (Nsmall\_trans \* zel1).flatten()

btemp2 = (Nsmall\_trans \* zel2).flatten()

btemp3 = (Nsmall\_trans \* zel3).flatten()

for n in range(len(Atemp)):

b1[con[i][n]-1] = b1[con[i][n]-1] + w[j]\*w[k]\* btemp1[n]\*J\_trans

b2[con[i][n]-1] = b2[con[i][n]-1] + w[j]\*w[k]\* btemp2[n]\*J\_trans

b3[con[i][n]-1] = b3[con[i][n]-1] + w[j]\*w[k]\* btemp3[n]\*J\_trans

for p in range(len(Atemp[0])):

A[con[i][n]-1][con[i][p]-1] = A[con[i][n]-1][con[i][p]-1] + J\_trans\*w[j]\*w[k]\*Atemp[n][p]

#Solve least squares

z\_neg1 = np.linalg.solve(A,b1)

z\_neg2 = np.linalg.solve(A,b2)

z\_neg3 = np.linalg.solve(A,b3)

#Grab the z on corresponding natural coords

centerNodes = np.array([[5,3],

[9,5]])

zcentera = [1,1]

zcenterb = [0,0]

zcenterc = [0,0]

for i in range(len(centerNodes)):

inel = whichel(centerNodes[i][0],centerNodes[i][1])

guess = np.array([0,0])

natcoords = sc.fsolve(func,guess,(gp\_global,el\_og[inel-1]))

ztemp = np.array([[zc[con[inel-1][0]-1],zc[con[inel-1][1]-1],zc[con[inel-1][2]-1],zc[con[inel-1][3]-1]]])

SFtemp,SFtempsmall = ShapeFcn(natcoords[0],natcoords[1])

zcenterb[i] = centerNodes[i][0]+2\*centerNodes[i][1]

zcenterc[i] = SFtempsmall @ ztemp.T

zcola = np.array([1,1,1,1,zcentera[0],1,1,zcentera[1],1,1,1,1])

zcolb = np.array([0,8,16,4,zcenterb[0],20,8,zcenterb[1],24,12,20,28])

zcolc = np.array([2,8,-7,6,zcenterc[0][0][0],1,3,zcenterc[1][0][0],4,-4,-3,0])

twonorm\_project\_a = 0

twonorm\_project\_b = 0

twonorm\_project\_c = 0

twonorm\_colloc\_a = 0

twonorm\_colloc\_b = 0

twonorm\_colloc\_c = 0

for i in range(numel):

gaussPoints = [-np.sqrt(3/5),0,np.sqrt(3/5)]

w = [5/9,8/9,5/9]

for j in range(len(w)):

for k in range(len(w)):

nodeVec\_og = np.array([[el\_og[i].NODE[0][0],el\_og[i].NODE[0][1],el\_og[i].NODE[1][0],el\_og[i].NODE[1][1],\

el\_og[i].NODE[2][0],el\_og[i].NODE[2][1],el\_og[i].NODE[3][0],el\_og[i].NODE[3][1]]])

nodeVec\_trans = np.array([[el\_trans[i].NODE[0][0],el\_trans[i].NODE[0][1],el\_trans[i].NODE[1][0],el\_trans[i].NODE[1][1],\

el\_trans[i].NODE[2][0],el\_trans[i].NODE[2][1],el\_trans[i].NODE[3][0],el\_trans[i].NODE[3][1]]])

N\_og, Nsmall\_og = ShapeFcn(gaussPoints[j],gaussPoints[k])

N\_trans, Nsmall\_trans = ShapeFcn(gaussPoints[j],gaussPoints[k])

globalGaussPointsx\_og.append((N\_og @ nodeVec\_og.T)[0])

globalGaussPointsy\_og.append((N\_og @ nodeVec\_og.T)[1])

globalGaussPointsx\_trans.append((N\_trans @ nodeVec\_trans.T)[0])

globalGaussPointsy\_trans.append((N\_trans @ nodeVec\_trans.T)[1])

J\_trans = np.linalg.det(Jacobian(gaussPoints[j],gaussPoints[k],el\_trans[i]))

gp\_global = [globalGaussPointsx\_trans[count][0],globalGaussPointsy\_trans[count][0]]

zel1\_true = 1

zel2\_true = globalGaussPointsx\_trans[count]+2\*globalGaussPointsy\_trans[count]

#which element are we in

inel = whichel(globalGaussPointsx\_trans[count],globalGaussPointsy\_trans[count])

#print(inel)

#find natural coords of gp

guess = np.array([0,0])

natcoords = sc.fsolve(func,guess,(gp\_global,el\_og[inel-1]))

#print(gp\_global)

#get interpolated values

ztemp = np.array([[zc[con[inel-1][0]-1],zc[con[inel-1][1]-1],zc[con[inel-1][2]-1],zc[con[inel-1][3]-1]]])

#print(inel)

#print(natcoords)

#print(ztemp)

SFtemp,SFtempsmall = ShapeFcn(natcoords[0],natcoords[1])

zel3\_true = SFtempsmall @ ztemp.T

#projected z at gauss point

ztempa = np.array([[z\_neg1[con[i][0]-1],z\_neg1[con[i][1]-1],z\_neg1[con[i][2]-1],z\_neg1[con[i][3]-1]]])

ztempb = np.array([[z\_neg2[con[i][0]-1],z\_neg2[con[i][1]-1],z\_neg2[con[i][2]-1],z\_neg2[con[i][3]-1]]])

ztempc = np.array([[z\_neg3[con[i][0]-1],z\_neg3[con[i][1]-1],z\_neg3[con[i][2]-1],z\_neg3[con[i][3]-1]]])

zgpap = (Nsmall\_trans @ ztempa.T)[0][0][0]

#print(zgpap)

zgpbp = (Nsmall\_trans @ ztempb.T)[0][0][0]

zgpcp = (Nsmall\_trans @ ztempc.T)[0][0][0]

#Collocated z at gauss point

ztempa = np.array([[zcola[con[i][0]-1],zcola[con[i][1]-1],zcola[con[i][2]-1],zcola[con[i][3]-1]]])

ztempb = np.array([[zcolb[con[i][0]-1],zcolb[con[i][1]-1],zcolb[con[i][2]-1],zcolb[con[i][3]-1]]])

ztempc = np.array([[zcolc[con[i][0]-1],zcolc[con[i][1]-1],zcolc[con[i][2]-1],zcolc[con[i][3]-1]]])

zgpac = (Nsmall\_trans @ ztempa.T)[0][0]

zgpbc = (Nsmall\_trans @ ztempb.T)[0][0]

zgpcc = (Nsmall\_trans @ ztempc.T)[0][0]

twonorm\_project\_a += ((zgpap-zel1\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_project\_b += ((zgpbp-zel2\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_project\_c += ((zgpcp-zel3\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_colloc\_a += ((zgpac-zel1\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_colloc\_b += ((zgpbc-zel2\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_colloc\_c += ((zgpcc-zel3\_true)\*\*2)\*w[j]\*w[k]\*J\_trans

twonorm\_project\_a = np.sqrt(twonorm\_project\_a)

twonorm\_project\_b = np.sqrt(twonorm\_project\_b)

twonorm\_project\_c = np.sqrt(twonorm\_project\_c)

twonorm\_colloc\_a = np.sqrt(twonorm\_colloc\_a)

twonorm\_colloc\_b = np.sqrt(twonorm\_colloc\_b)

twonorm\_colloc\_b = np.sqrt(twonorm\_colloc\_b)

# zmesh1og = np.hstack(zmesh1og)

# zmesh2og = np.hstack(zmesh2og)

# zmesh3og = np.hstack(zmesh3og)

# zmesh1trans = np.hstack(zmesh1trans).flatten()

# zmesh2trans = np.hstack(zmesh2trans).flatten()

# zmesh3trans = np.hstack(zmesh3trans).flatten()

# x1 = np.hstack(x1)

# y1 = np.hstack(y1)

# ax = plt.axes(projection = '3d')

# #x1 = np.array(x1)

# #x2 = np.array(x2)

# ax.plot\_trisurf(x1,y1,zmesh2og)

# ax.plot\_trisurf(x2,y2,zmesh2trans)

#plt.plot(x1,y1,'o')

#plt.plot(x2,y2,'o')

plotElements(nodesnat,con,'b')

#plt.plot(globalGaussPointsx\_og,globalGaussPointsy\_og,'o')

#plt.figure(2)

#plotElements(nodes,con,'b')

#plt.plot(globalGaussPointsx\_trans,globalGaussPointsy\_trans,'o')