```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May  6 10:14:47 2022

@author: juanmeriles
"""
import numpy as np
import matplotlib.pyplot as plt

class element:
    NODE = []
    CON = []
    BOUN = []
    id_v = []

    def __init__(self):
        pass

def Shape(eta):
    N1 = 1-eta
    N2 = eta
    N = np.array([[N1,0,N2,0],
                  [0,N1,0,N2]])
    return N

def function(elcoords,L):
    x0 = elcoords[0]
    y0 = elcoords[1]
    x1 = elcoords[2]
    y1 = elcoords[3]
    fe = (1/(2*L*np.sqrt((x1-x0)**2+(y1-y0)**2)))*np.array([[-(x1-x0)*(y0+y1)],
                   [(x0-x1)**2+2*y0*(y0-y1)],
                   [(x1-x0)*(y0+y1)],
                   [(x0-x1)**2+2*y1*(y1-y0)]])
    return fe

def elstiff(elcoords,L):
    x0 = elcoords[0]
    y0 = elcoords[1]
    x1 = elcoords[2]
    y1 = elcoords[3]
    k11 = (y0+y1)*(y0-y1)**2
    k12 = (x0-x1)*((x0-x1)**2+2*y1*(y1-y0))
    k13 = -(y0+y1)*(y0-y1)**2
    k14 = (x0-x1)*((x0-x1)**2+2*y0*(y0-y1))
    k22 = (x0**2)*(3*y0-y1)+2*x0*x1*(y1-3*y0)+(x1**2)*(3*y0-y1)+2*(y0-y1)**3
    k23 = -(x0-x1)*((x0-x1)**2+2*y1*(y1-y0))
```

```
    k24 = -((x0-x1)**2)*(y0+y1)
    k33 = (y0+y1)*(y0-y1)**2
    k34 = -(x0-x1)*((x0-x1)**2+2*y0*(y0-y1))
    k44 = -(x0**2)*(y0-3*y1)+2*x0*x1*(y0-3*y1)-x1**2*(y0-3*y1)-2*(y0-y1)**3
    C = 1/(2*L*(((x1-x0)**2)+(y1-y0)**2)**(3/2))
    ke = C*np.array([[k11,k12,k13,k14],
                     [k12,k22,k23,k24],
                     [k13,k23,k33,k34],
                     [k14,k24,k34,k44]])
    return ke

def dmat(elcoords,L):
    x0 = elcoords[0]
    y0 = elcoords[1]
    x1 = elcoords[2]
    y1 = elcoords[3]
    C = 1/((((x1-x0)**2)+(y1-y0)**2)**(1/2))
    de = C*np.array([[x0-x1],
                     [y0-y1],
                     [x1-x0],
                     [y1-y0]])
    return de

def ddmat(elcoords,L):
    x0 = elcoords[0]
    y0 = elcoords[1]
    x1 = elcoords[2]
    y1 = elcoords[3]
    d11 = (y0-y1)**2
    d12 = -(x0-x1)*(y0-y1)
    d13 = -(y0-y1)**2
    d14 = (x0-x1)*(y0-y1)
    d22 = (x0-x1)**2
    d23 = (x0-x1)*(y0-y1)
    d24 = -(x0-x1)**2
    d33 = (y1-y0)**2
    d34 = (x1-x0)*(y0-y1)
    d44 = (x1-x0)**2
    C = 1/((((x1-x0)**2)+(y1-y0)**2)**(3/2))
    dde = C*np.array([[d11,d12,d13,d14],
                      [d12,d22,d23,d24],
                      [d13,d23,d33,d34],
                      [d14,d24,d34,d44]])
    return dde


r = .306392
ne = 16
```

```
maxit = 1000
L = 4
a = 1
slope = L/a
xexact = np.arange(-a,a,.01)
yexact = r*np.cosh(xexact/r)-np.sqrt((r**2)+(L**2))
x = np.arange(0,a+a/ne,a/ne)
y = slope*x-a*slope
U = []
nodes = []
con = []
boun = []
elid = []
globaldofs = []
for i in range(len(x)-1):
    nodes.append([x[i],y[i]])
    con.append([i,i+1])
    boun.append([0,0])
    globaldofs.append([2*i,2*i+1])
    elid.append([2*i,2*i+1,2*i+2,2*i+3])
    U.append([x[i],y[i]])


i=i+1
globaldofs.append([2*i,2*i+1])
boun.append([0,0])
nodes.append([x[i],y[i]])
lam = np.zeros((ne))
U.append([x[i],y[i]])
U = np.array(U).flatten()

globaldofs = np.array(globaldofs)
globaldofs = globaldofs.flatten()


elements = []
boun[0] = [1,0]
boun[-1] = [1,1]

count = 0
BDOFS = []
FDOFS = []
for i in range(len(boun)):
    for j in range(len(boun[0])):
        if boun[i][j] == 1:
            BDOFS.append(count)
        else:
            FDOFS.append(count)
```

```
        count = count+1

globaldofs = np.hstack([globaldofs[FDOFS],globaldofs[BDOFS]])

#Creates the ReDOF and RepDOF vectors which map the old global dofs to new positi
DOForder = np.hstack([FDOFS,BDOFS])
ReDOFS = [0]*(2*len(nodes))
for i in range(len(DOForder)):
    ReDOFS[DOForder[i]] = i

for i in range(len(con)):
    elements.insert(i,element())
    elements[i].NODE = np.array([nodes[con[i][0]],nodes[con[i][1]]])
    elements[i].CON = con[i]
    elements[i].BOUN = np.array([boun[con[i][0]],boun[con[i][1]]])
    elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0][1]
    elements[i].id_v = elid[i]
    elements[i].ke = elstiff(elements[i].nodeVec,L)
    elements[i].fe = function(elements[i].nodeVec,L)
    elements[i].de = dmat(elements[i].nodeVec,L)
    elements[i].dde = ddmat(elements[i].nodeVec,L)
    elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**2+(e

#assembly
K = np.zeros((2*len(nodes),2*len(nodes)))
Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
f = np.zeros((2*len(nodes)))
d = np.zeros((ne,2*len(nodes)))
c = np.zeros((ne))
Duold = np.zeros(len(FDOFS)+ne)

for i in range(ne):
    c[i] += elements[i].L-L/ne
    for j in range(len(elements[i].fe)):
        f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
        d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
        for k in range(len(elements[i].fe)):
            ind1 = ReDOFS[elements[i].id_v[j]]
            ind2 = ReDOFS[elements[i].id_v[k]]
            K[ind1,ind2] += elements[i].ke[j,k]
            Dd[ind1,ind2,i] += elements[i].dde[j,k]

Kff = K[0:len(FDOFS),0:len(FDOFS)]
Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
ff = f[0:len(FDOFS)]
df = d[:,0:len(FDOFS)]
Uf = U[FDOFS]
```

```
d = d.T
df = df.T
err = 100
tol = 10**-6
A = np.block([[Kff + np.tensordot(Ddff, lam, axes=([2,0])),df], [df.T, np.zeros((
b = -np.hstack((ff+df@lam,c))
b = b.T
Du = np.linalg.solve(A,b)

err = np.linalg.norm(Du-Duold,2)
it = 0

while err>tol and it<maxit:
    lam = lam+Du[len(FDOFS):]
    Du = Du[0:len(FDOFS)]
    Du = np.hstack([Du,np.zeros(len(BDOFS))])
    U = U+Du[ReDOFS]
    U = np.reshape(U,(int(len(U)/2),2))

    for i in range(len(con)):
        elements[i].NODE = np.array([U[con[i][0]],U[con[i][1]]])
        elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0
        elements[i].ke = elstiff(elements[i].nodeVec,L)
        elements[i].fe = function(elements[i].nodeVec,L)
        elements[i].de = dmat(elements[i].nodeVec,L)
        elements[i].dde = ddmat(elements[i].nodeVec,L)
        elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**

    #assembly
    U=U.flatten()
    K = np.zeros((2*len(nodes),2*len(nodes)))
    Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
    f = np.zeros((2*len(nodes)))
    d = np.zeros((ne,2*len(nodes)))
    c = np.zeros((ne))

    for i in range(ne):
        c[i] += elements[i].L-L/ne
        for j in range(len(elements[i].fe)):
            f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
            d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
            for k in range(len(elements[i].fe)):
                ind1 = ReDOFS[elements[i].id_v[j]]
                ind2 = ReDOFS[elements[i].id_v[k]]
                K[ind1,ind2] += elements[i].ke[j,k]
                Dd[ind1,ind2,i] += elements[i].dde[j,k]
```

```
    Kff = K[0:len(FDOFS),0:len(FDOFS)]
    Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
    ff = f[0:len(FDOFS)]
    df = d[:,0:len(FDOFS)]

    d = d.T
    df = df.T
    A = np.block([[Kff + np.tensordot(Ddff, lam, axes=([2,0])),df], [df.T, np.zer
    b = -np.hstack((ff+df@lam,c))
    b = b.T
    Du = np.linalg.solve(A,b)

    err = np.linalg.norm(Du-Duold,2)
    Duold = Du
    it = it+1

U = np.reshape(U,(int(len(U)/2),2))
plt.plot(xexact,yexact,label='True Solution')
plt.plot(-U[:,0],U[:,1],'r',label = 'Classical Multipliers')
plt.plot(U[:,0],U[:,1],'r')
plt.legend()
plt.title('True Solution vs Classical Multipliers')
it1 = it
err1 = err



#classical Penalty
#Reset evertything

x = np.arange(0,a+a/ne,a/ne)
y = slope*x-a*slope
U = []
nodes = []
con = []
boun = []
elid = []
globaldofs = []
for i in range(len(x)-1):
    nodes.append([x[i],y[i]])
    con.append([i,i+1])
    boun.append([0,0])
    globaldofs.append([2*i,2*i+1])
    elid.append([2*i,2*i+1,2*i+2,2*i+3])
    U.append([x[i],y[i]])
```

```
i=i+1
globaldofs.append([2*i,2*i+1])
boun.append([0,0])
nodes.append([x[i],y[i]])
lam = np.zeros((ne))
U.append([x[i],y[i]])
U = np.array(U).flatten()

globaldofs = np.array(globaldofs)
globaldofs = globaldofs.flatten()


elements = []
boun[0] = [1,0]
boun[-1] = [1,1]

count = 0
BDOFS = []
FDOFS = []
for i in range(len(boun)):
    for j in range(len(boun[0])):
        if boun[i][j] == 1:
            BDOFS.append(count)
        else:
            FDOFS.append(count)
        count = count+1

globaldofs = np.hstack([globaldofs[FDOFS],globaldofs[BDOFS]])

#Creates the ReDOF and RepDOF vectors which map the old global dofs to new positi
DOForder = np.hstack([FDOFS,BDOFS])
ReDOFS = [0]*(2*len(nodes))
for i in range(len(DOForder)):
    ReDOFS[DOForder[i]] = i

for i in range(len(con)):
    elements.insert(i,element())
    elements[i].NODE = np.array([nodes[con[i][0]],nodes[con[i][1]]])
    elements[i].CON = con[i]
    elements[i].BOUN = np.array([boun[con[i][0]],boun[con[i][1]]])
    elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0][1]
    elements[i].id_v = elid[i]
    elements[i].ke = elstiff(elements[i].nodeVec,L)
    elements[i].fe = function(elements[i].nodeVec,L)
    elements[i].de = dmat(elements[i].nodeVec,L)
    elements[i].dde = ddmat(elements[i].nodeVec,L)
    elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**2+(e
```

```
#assembly
K = np.zeros((2*len(nodes),2*len(nodes)))
Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
f = np.zeros((2*len(nodes)))
d = np.zeros((ne,2*len(nodes)))
c = np.zeros((ne))
Duold = np.zeros(len(FDOFS))

for i in range(ne):
    c[i] += elements[i].L-L/ne
    for j in range(len(elements[i].fe)):
        f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
        d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
        for k in range(len(elements[i].fe)):
            ind1 = ReDOFS[elements[i].id_v[j]]
            ind2 = ReDOFS[elements[i].id_v[k]]
            K[ind1,ind2] += elements[i].ke[j,k]
            Dd[ind1,ind2,i] += elements[i].dde[j,k]

Kff = K[0:len(FDOFS),0:len(FDOFS)]
Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
ff = f[0:len(FDOFS)]
df = d[:,0:len(FDOFS)]
Uf = U[FDOFS]
eps = 10**3

d = d.T
df = df.T
err = 100
tol = 10**-6
A = Kff+eps*(np.tensordot(Ddff, c, axes=([2,0]))+df@df.T)
b = -(ff+eps*df@c)
b = b.T
Du = np.linalg.solve(A,b)

err = np.linalg.norm(Du-Duold,2)
it = 0


while err>tol and it<maxit:
    Du = np.hstack([Du,np.zeros(len(BDOFS))])
    U = U+Du[ReDOFS]
    U = np.reshape(U,(int(len(U)/2),2))

    for i in range(len(con)):
        elements[i].NODE = np.array([U[con[i][0]],U[con[i][1]]])
        elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0
        elements[i].ke = elstiff(elements[i].nodeVec,L)
```

```
        elements[i].fe = function(elements[i].nodeVec,L)
        elements[i].de = dmat(elements[i].nodeVec,L)
        elements[i].dde = ddmat(elements[i].nodeVec,L)
        elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**

    #assembly
    U=U.flatten()
    K = np.zeros((2*len(nodes),2*len(nodes)))
    Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
    f = np.zeros((2*len(nodes)))
    d = np.zeros((ne,2*len(nodes)))
    c = np.zeros((ne))

    for i in range(ne):
        c[i] += elements[i].L-L/ne
        for j in range(len(elements[i].fe)):
            f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
            d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
            for k in range(len(elements[i].fe)):
                ind1 = ReDOFS[elements[i].id_v[j]]
                ind2 = ReDOFS[elements[i].id_v[k]]
                K[ind1,ind2] += elements[i].ke[j,k]
                Dd[ind1,ind2,i] += elements[i].dde[j,k]

    Kff = K[0:len(FDOFS),0:len(FDOFS)]
    Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
    ff = f[0:len(FDOFS)]
    df = d[:,0:len(FDOFS)]

    d = d.T
    df = df.T
    A = Kff+eps*(np.tensordot(Ddff, c, axes=([2,0]))+df@df.T)
    b = -(ff+eps*df@c)
    b = b.T
    Du = np.linalg.solve(A,b)

    err = np.linalg.norm(Du-Duold,2)
    Duold = Du
    it = it+1

U = np.reshape(U,(int(len(U)/2),2))
plt.figure(10)
plt.plot(xexact,yexact,label='True Solution')
plt.plot(-U[:,0],U[:,1],'r',label = 'Classical Penalty')
plt.plot(U[:,0],U[:,1],'r')
plt.legend()
plt.title('True Solution vs Classical Penalty')
it2 = it
```

```
err2 = err

#augmented
#Reset evertything

x = np.arange(0,a+a/ne,a/ne)
y = slope*x-a*slope
U = []
nodes = []
con = []
boun = []
elid = []
globaldofs = []
for i in range(len(x)-1):
    nodes.append([x[i],y[i]])
    con.append([i,i+1])
    boun.append([0,0])
    globaldofs.append([2*i,2*i+1])
    elid.append([2*i,2*i+1,2*i+2,2*i+3])
    U.append([x[i],y[i]])


i=i+1
globaldofs.append([2*i,2*i+1])
boun.append([0,0])
nodes.append([x[i],y[i]])
lam = np.zeros((ne))
U.append([x[i],y[i]])
U = np.array(U).flatten()

globaldofs = np.array(globaldofs)
globaldofs = globaldofs.flatten()


elements = []
boun[0] = [1,0]
boun[-1] = [1,1]

count = 0
BDOFS = []
FDOFS = []
for i in range(len(boun)):
    for j in range(len(boun[0])):
        if boun[i][j] == 1:
            BDOFS.append(count)
        else:
            FDOFS.append(count)
        count = count+1
```

```
globaldofs = np.hstack([globaldofs[FDOFS],globaldofs[BDOFS]])

#Creates the ReDOF and RepDOF vectors which map the old global dofs to new positi
DOForder = np.hstack([FDOFS,BDOFS])
ReDOFS = [0]*(2*len(nodes))
for i in range(len(DOForder)):
    ReDOFS[DOForder[i]] = i

for i in range(len(con)):
    elements.insert(i,element())
    elements[i].NODE = np.array([nodes[con[i][0]],nodes[con[i][1]]])
    elements[i].CON = con[i]
    elements[i].BOUN = np.array([boun[con[i][0]],boun[con[i][1]]])
    elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0][1]
    elements[i].id_v = elid[i]
    elements[i].ke = elstiff(elements[i].nodeVec,L)
    elements[i].fe = function(elements[i].nodeVec,L)
    elements[i].de = dmat(elements[i].nodeVec,L)
    elements[i].dde = ddmat(elements[i].nodeVec,L)
    elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**2+(e

#assembly
K = np.zeros((2*len(nodes),2*len(nodes)))
Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
f = np.zeros((2*len(nodes)))
d = np.zeros((ne,2*len(nodes)))
c = np.zeros((ne))
Duold = np.zeros(len(FDOFS))

for i in range(ne):
    c[i] += elements[i].L-L/ne
    for j in range(len(elements[i].fe)):
        f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
        d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
        for k in range(len(elements[i].fe)):
            ind1 = ReDOFS[elements[i].id_v[j]]
            ind2 = ReDOFS[elements[i].id_v[k]]
            K[ind1,ind2] += elements[i].ke[j,k]
            Dd[ind1,ind2,i] += elements[i].dde[j,k]

Kff = K[0:len(FDOFS),0:len(FDOFS)]
Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
ff = f[0:len(FDOFS)]
df = d[:,0:len(FDOFS)]
Uf = U[FDOFS]
eps = 10**3
zeta = eps
```

```
d = d.T
df = df.T
err = 100
tol = 10**-6
A = Kff + np.tensordot(Ddff, lam+eps*c, axes=([2,0])) + eps*df@df.T
b = -(ff+eps*df@c)
b = b.T
Du = np.linalg.solve(A,b)

err = np.linalg.norm(Du-Duold,2)
it = 0

while err>tol and it <maxit:
    Du = np.hstack([Du,np.zeros(len(BDOFS))])
    lam = zeta*c
    U = U+Du[ReDOFS]
    U = np.reshape(U,(int(len(U)/2),2))

    for i in range(len(con)):
        elements[i].NODE = np.array([U[con[i][0]],U[con[i][1]]])
        elements[i].nodeVec = np.array([elements[i].NODE[0][0],elements[i].NODE[0
        elements[i].ke = elstiff(elements[i].nodeVec,L)
        elements[i].fe = function(elements[i].nodeVec,L)
        elements[i].de = dmat(elements[i].nodeVec,L)
        elements[i].dde = ddmat(elements[i].nodeVec,L)
        elements[i].L = np.sqrt((elements[i].NODE[1][0]-elements[i].NODE[0][0])**

    #assembly
    U=U.flatten()
    K = np.zeros((2*len(nodes),2*len(nodes)))
    Dd = np.zeros((2*len(nodes),2*len(nodes),ne))
    f = np.zeros((2*len(nodes)))
    d = np.zeros((ne,2*len(nodes)))
    c = np.zeros((ne))

    for i in range(ne):
        c[i] += elements[i].L-L/ne
        for j in range(len(elements[i].fe)):
            f[ReDOFS[elements[i].id_v[j]]] += elements[i].fe[j]
            d[i,ReDOFS[elements[i].id_v[j]]] += elements[i].de[j]
            for k in range(len(elements[i].fe)):
                ind1 = ReDOFS[elements[i].id_v[j]]
                ind2 = ReDOFS[elements[i].id_v[k]]
                K[ind1,ind2] += elements[i].ke[j,k]
                Dd[ind1,ind2,i] += elements[i].dde[j,k]

    Kff = K[0:len(FDOFS),0:len(FDOFS)]
```

```
    Ddff = Dd[0:len(FDOFS),0:len(FDOFS)]
    ff = f[0:len(FDOFS)]
    df = d[:,0:len(FDOFS)]

    d = d.T
    df = df.T
    A = Kff + np.tensordot(Ddff, lam+eps*c, axes=([2,0])) + eps*df@df.T
    b = -(ff+eps*df@c)
    b = b.T
    Du = np.linalg.solve(A,b)

    err = np.linalg.norm(Du-Duold,2)
    Duold = Du
    it = it+1

U = np.reshape(U,(int(len(U)/2),2))
plt.figure(39)
plt.plot(xexact,yexact,label='True Solution')
plt.plot(-U[:,0],U[:,1],'r',label = 'Augmented Lagrangian')
plt.plot(U[:,0],U[:,1],'r')
plt.legend()
plt.title('True Solution vs Augmented Lagrangian')
it3 = it
err3 = err
```