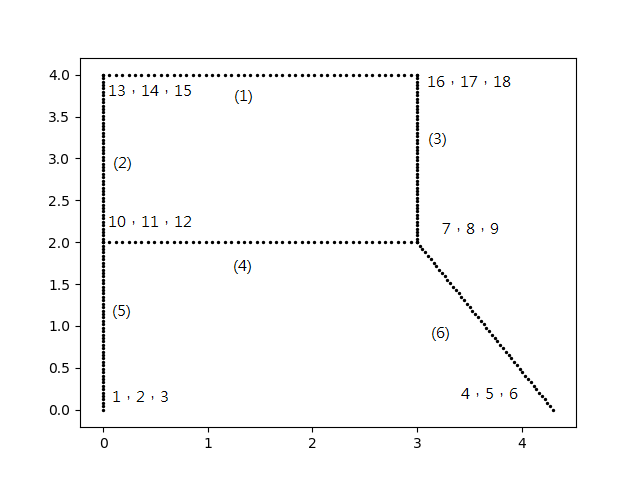
有限元素project2

學號:4105061124

姓名:謝昉澂

定義:

****

# **Question1:** orientations and stiff matrix of Bar 5 in global coordinates

Orientations of bar 5 is 90 degree。

stiff matrix of Bar 5 in global coordinates

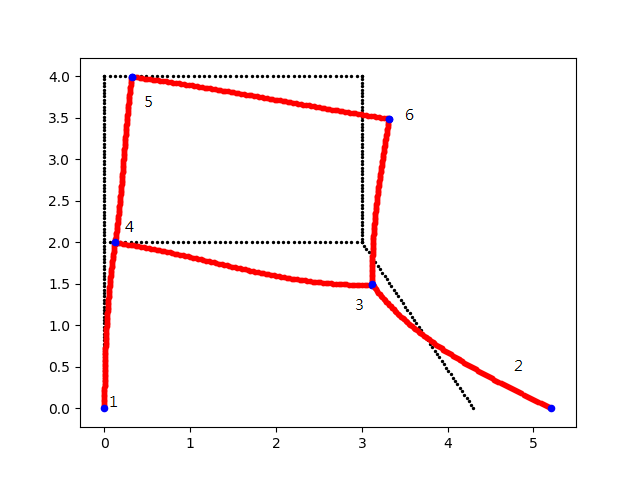
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1.02e+08 | 1.73e+02 | -1.02e+08 | -1.02e+08 | -1.73e+02 | -1.02e+08 |
| 1.73e+02 | 6.56e+09 | 2.74e+00 | -1.73e+02 | -6.56e+09 | 2.74e+00 |
| -1.02e+08 | 2.74e+00 | 1.36e+08 | 1.02e+08 | -2.74e+00 | 6.83e+07 |
| -1.02e+08 | -1.73e+02 | 1.02e+08 | 1.02e+08 | 1.73e+02 | 1.02e+08 |
| -1.73e+02 | -6.56e+09 | -2.74e+00 | 1.73e+02 | 6.56e+09 | -2.74e+00 |
| -1.02e+08 | 2.74e+00 | 6.83e+07 | 1.02e+08 | -2.74e+00 | 1.36e+08 |

# **Question2:** self-weight vectors for each bar.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| bar | Global\_fx1 | Global\_fy1 | Global\_m1 | Global\_fx2 | Global\_fy2 | Global\_m2 |
| 1 | 0 | -7.26e+03 | -3.63e+03 | 0 | -7.26e+03 | 3.63e+03 |
| 2 | 0 | -4.84e+03 | -4.32e-05 | 0 | -4.84e+03 | 4.32e-05 |
| 3 | 0 | -4.84e+03 | -4.32e-05 | 0 | -4.84e+03 | 4.32e-05 |
| 4 | 0 | -7.26e+03 | -3.63e+03 | 0 | -7.26e+03 | 3.63e+03 |
| 5 | 0 | -4.84e+03 | -4.32e-05 | 0 | -4.84e+03 | 4.32e-05 |
| 6 | 0 | -5.77e+03 | 1.25e+03 | 0 | -5.77e+03 | -1.25e+03 |

# **Question3:** nodal deflection at all nodes

圖中變形量為實際上的1000倍。

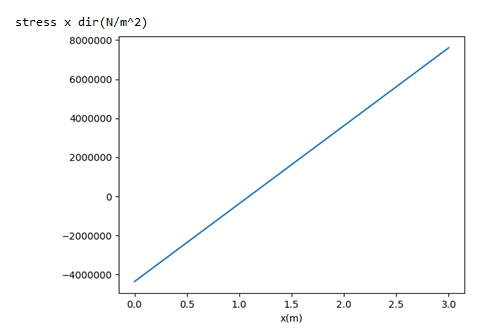


|  |  |  |  |
| --- | --- | --- | --- |
| node | Deflection x | Deflection y | Deflection θ |
| 1 | 0.000e+00 | 0.000e+00 | 0.000e+00 |
| 2 | 9.068e-04 | 0.000e+00 | 5.830e-04 |
| 3 | 1.241e-04 | -5.127e-04 | 3.256e-05 |
| 4 | 1.228e-04 | -6.354e-06 | -1.423e-04 |
| 5 | 3.184e-04 | -8.543e-06 | -1.312e-04 |
| 6 | 3.166e-04 | -5.150e-04 | -1.569e-04 |

# **Question4: determine the maxima local axial stress in bar4**

Discussion:

共考慮moment造成的軸向應力，和x方向變形造成的軸向應力。

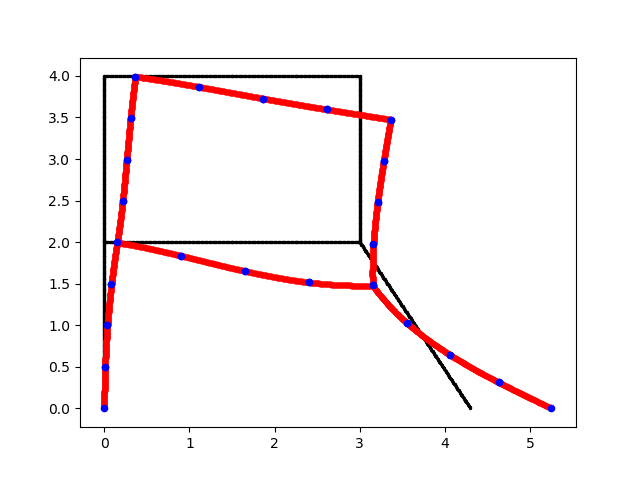


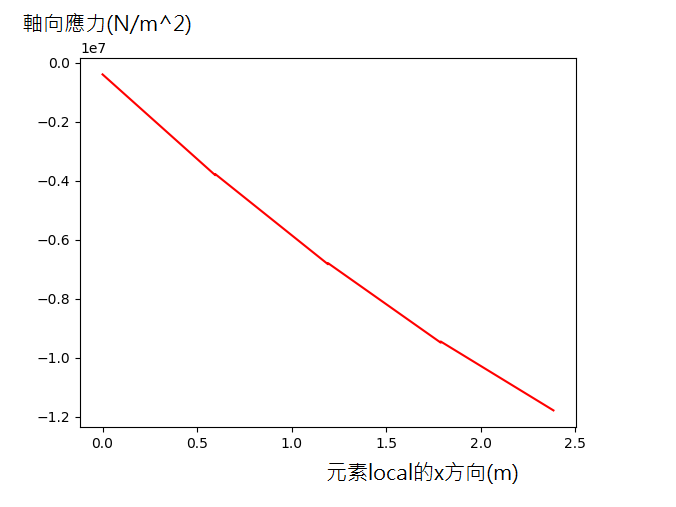
軸向應力最大值：7607603(N/m^2)

# **Question5: the value of d such that the maximum axial stress in bar 6 is minimized**

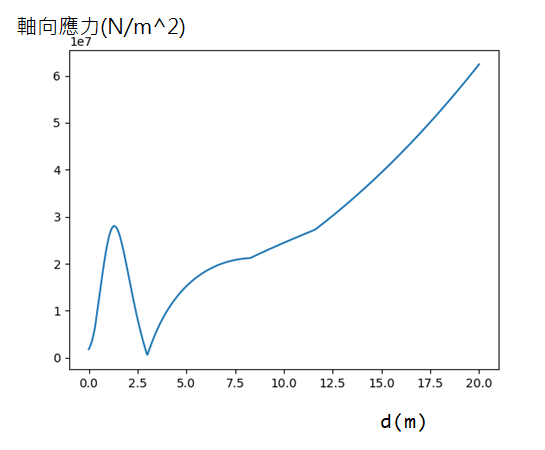
在此題中我把每個元素再細切成四段。

先用程式算出元素local的x方向軸應力，再取其中絕對值最大值，當作最大軸向應力的值，下圖為d=4.3m的情況，最大軸向應力的值為1.178e7(N/m^2)。



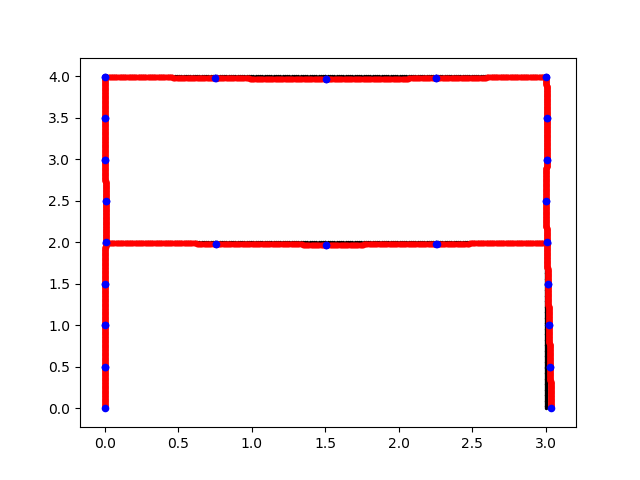


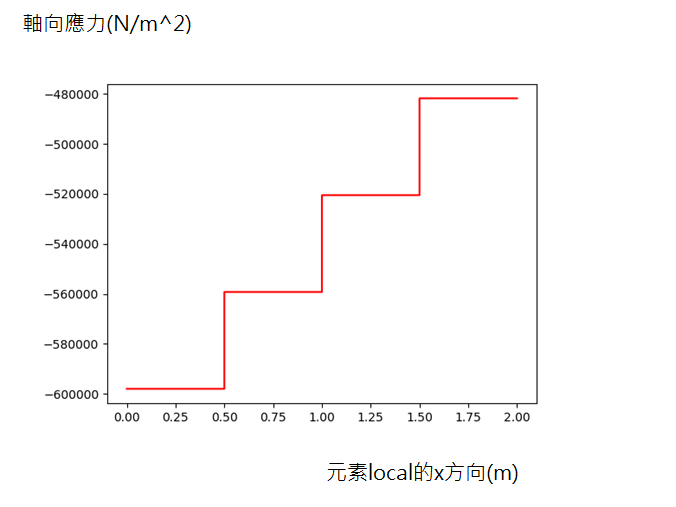
我將距離d從0到20以等間距500次分析，計算最大軸向應力的值如下圖



最小軸向應力值為618566N/m^2

此時d=3m

結構如下圖。



Code：

Main程式是用來解還沒有細化元素的問題。

Main2程式是用來解細化元素的問題。

Change\_d\_main程式是用來解第5題專用的。

其他程式都是副程式。

Main.py

import numpy as np  
from element import get\_vecA, get\_vecL, get\_vecE, get\_vecTheta, get\_density, get\_ele\_coor  
from node import node\_member  
from connectivity\_matrix import get\_mtxEFT  
from global\_stiff\_matrix import get\_mtx\_K\_glo  
from add\_BC import mtx\_K\_glo\_add\_BC  
from add\_BC import vecf\_add\_BC  
from show\_picture import show\_picture  
from show\_picture import beam\_scatter  
from distribute\_force import self\_weight\_ele\_to\_node  
from distribute\_force import add\_q\_force\_ele\_to\_node  
from analy\_shearMoment import analy\_shearMoment  
from refine\_mesh\_para import refine\_mesh\_para  
import matplotlib.pyplot as plt  
  
# exaggerating deformation  
u\_mul = 1000  
  
  
def main():  
 # get element and node data  
 vecA = get\_vecA()  
 vecL = get\_vecL()  
 vecE = get\_vecE()  
 vecTheta = get\_vecTheta()  
 density = get\_density()  
 N\_gdof, vecFix, vecF = node\_member()  
 mtxEFT = get\_mtxEFT()  
 N\_e = mtxEFT.shape[0]  
  
 # add force  
 vecF[17 - 1] += -5000  
  
 q\_force\_global = np.array([-1000.0, 0, 0])  
 vecF += add\_q\_force\_ele\_to\_node(mtxEFT, 3, q\_force\_global, vecL, vecTheta)  
 # add self\_weight  
 for N\_e\_i in range(N\_e):  
 vecF\_weight = self\_weight\_ele\_to\_node(mtxEFT, N\_e\_i + 1, vecA, vecL, vecTheta, density)  
 vecF += vecF\_weight  
  
 # create stiff matrix  
 mtx\_K\_glo = get\_mtx\_K\_glo(mtxEFT, N\_gdof, N\_e, vecE, vecA, vecL, vecTheta)  
  
 # add boundary condition  
 mtx\_K\_glo\_added\_BC = mtx\_K\_glo\_add\_BC(mtx\_K\_glo, vecFix)  
 vecF\_added\_bc = vecf\_add\_BC(vecF, vecFix)  
  
 # calculate vecU  
 inv\_k = np.linalg.inv(mtx\_K\_glo\_added\_BC)  
 vecU = inv\_k @ vecF\_added\_bc  
  
 # post process calculate external-force  
 F\_ext = mtx\_K\_glo @ vecU  
  
 # get external-force exert on element  
 # get element deformation  
 ele\_f\_ext = np.zeros([N\_e, 6])  
 ele\_u = np.zeros([N\_e, 6])  
 ele\_u\_no = mtxEFT  
 for i in range(N\_e):  
 for j in range(6):  
 ele\_u[i][j] = vecU[ele\_u\_no[i][j] - 1]  
 ele\_f\_ext[i][j] = F\_ext[ele\_u\_no[i][j] - 1]  
  
 # analysis shear-force & moment & stress of special element  
 element\_no = 4  
 xs, ys\_stress\_xdir\_c\_pos, ys\_stress\_xdir\_c\_neg = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1],  
 vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1],  
 plot\_picture=False)  
 if (np.max(np.abs(ys\_stress\_xdir\_c\_pos)) > np.max(np.abs(ys\_stress\_xdir\_c\_neg))):  
 ys\_stress\_max = ys\_stress\_xdir\_c\_pos  
 else:  
 ys\_stress\_max = ys\_stress\_xdir\_c\_neg  
 plt.plot(xs, ys\_stress\_max)  
 plt.xlabel('x(m)')  
 plt.ylabel('stress x dir(N/m^2)')  
 plt.show()  
 print(np.max(np.abs(ys\_stress\_max)))  
 # show origin structure  
 ele\_coor = get\_ele\_coor()  
 show\_picture(ele\_coor, 'k')  
  
 # show deformation structure  
 ele\_u\_mul = ele\_u \* u\_mul  
 for e\_i in range(N\_e):  
 ele\_i\_ori = ele\_coor[e\_i] + np.array(  
 [ele\_u\_mul[e\_i][0], ele\_u\_mul[e\_i][1], ele\_u\_mul[e\_i][3], ele\_u\_mul[e\_i][4]])  
 ele\_i\_L = vecL[e\_i]  
 ele\_i\_the = vecTheta[e\_i]  
 xs, ys = beam\_scatter(ele\_i\_ori, ele\_i\_L, ele\_i\_the,  
 ele\_u\_mul[e\_i])  
 plt.scatter(xs, ys, c='r', s=10)  
 plt.scatter(xs[0], ys[0], c='b', s=20)  
 plt.scatter(xs[-1], ys[-1], c='b', s=20)  
 plt.show()  
  
 pass  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

main2.py

import numpy as np  
from element import get\_vecA, get\_vecL, get\_vecE, get\_vecTheta, get\_density, get\_ele\_coor  
from element import get\_vecL\_from\_coor, get\_vecTheta\_from\_coor  
from node import node\_member  
from connectivity\_matrix import get\_mtxEFT  
from global\_stiff\_matrix import get\_mtx\_K\_glo  
from add\_BC import mtx\_K\_glo\_add\_BC  
from add\_BC import vecf\_add\_BC  
from show\_picture import show\_picture  
from show\_picture import beam\_scatter  
from distribute\_force import self\_weight\_ele\_to\_node  
from distribute\_force import add\_q\_force\_ele\_to\_node  
from analy\_shearMoment import analy\_shearMoment  
from refine\_mesh\_para import refine\_mesh\_para  
from refine\_mesh\_para import refine\_mesh\_coor  
from refine\_mesh\_para import record\_element\_NO  
import matplotlib.pyplot as plt  
  
# exaggerating deformation  
u\_mul = 1000  
  
  
def main2():  
 # get element and node data  
 ele\_coor = get\_ele\_coor()  
 vecL = get\_vecL\_from\_coor(ele\_coor)  
 vecTheta = get\_vecTheta\_from\_coor(ele\_coor)  
 vecA = get\_vecA()  
 vecE = get\_vecE()  
 density = get\_density()  
 N\_gdof, vecFix, vecF = node\_member()  
 mtxEFT = get\_mtxEFT()  
  
 # record element to analysis  
 record\_element1 = [1]  
 record\_element2 = [2]  
 record\_element3 = [3]  
 record\_element4 = [4]  
 record\_element5 = [5]  
 record\_element6 = [6]  
  
 # refine element  
  
 element\_number = 6  
 refine\_element\_list = np.arange(1, element\_number + 1)  
 refine\_times = 1  
 for i in range(1,refine\_times+1):  
 element\_number = element\_number\*2  
 refine\_element\_list = np.hstack([refine\_element\_list, np.arange(1, element\_number+1)])  
  
 for refine\_element\_NO in refine\_element\_list:  
 mtxEFT, vecE, vecA, vecL, vecTheta = refine\_mesh\_para(mtxEFT, vecE, vecA, vecL, vecTheta, refine\_element\_NO)  
 ele\_coor = refine\_mesh\_coor(ele\_coor, refine\_element\_NO)  
 # adjust F distribute  
 # Zeroing force and add force  
 vecF = np.zeros([np.max(mtxEFT)])  
 vecF[17 - 1] += -5000  
 q\_force\_global = np.array([-1000.0, 0, 0])  
 # record\_q\_force should exert on which elements  
 record\_q\_force\_element3 = np.array([3])  
 if refine\_element\_NO in record\_q\_force\_element3:  
 record\_q\_force\_element3 = np.hstack([record\_q\_force\_element3, refine\_element\_NO])  
 for q\_force\_element\_i in record\_q\_force\_element3:  
 vecF += add\_q\_force\_ele\_to\_node(mtxEFT, q\_force\_element\_i, q\_force\_global, vecL, vecTheta)  
  
 # add self\_weight  
 N\_e = mtxEFT.shape[0]  
 for N\_e\_i in range(N\_e):  
 vecF\_weight = self\_weight\_ele\_to\_node(mtxEFT, N\_e\_i + 1, vecA, vecL, vecTheta, density)  
 vecF += vecF\_weight  
  
 # create stiff matrix  
 mtx\_K\_glo = get\_mtx\_K\_glo(mtxEFT, np.max(mtxEFT), N\_e, vecE, vecA, vecL, vecTheta)  
  
 # add boundary condition  
 mtx\_K\_glo\_added\_BC = mtx\_K\_glo\_add\_BC(mtx\_K\_glo, vecFix)  
 vecF\_added\_bc = vecf\_add\_BC(vecF, vecFix)  
  
 # calculate vecU  
 inv\_k = np.linalg.inv(mtx\_K\_glo\_added\_BC)  
 vecU = inv\_k @ vecF\_added\_bc  
  
 # post process calculate external-force  
 F\_ext = mtx\_K\_glo @ vecU  
  
 # get external-force exert on element  
 # get element deformation  
 ele\_f\_ext = np.zeros([N\_e, 6])  
 ele\_u = np.zeros([N\_e, 6])  
 ele\_u\_no = mtxEFT  
 for i in range(N\_e):  
 for j in range(6):  
 ele\_u[i][j] = vecU[ele\_u\_no[i][j] - 1]  
 ele\_f\_ext[i][j] = F\_ext[ele\_u\_no[i][j] - 1]  
  
 # record element to analysis  
 record\_element1 = record\_element\_NO(record\_element1, refine\_element\_NO, N\_e)  
 record\_element2 = record\_element\_NO(record\_element2, refine\_element\_NO, N\_e)  
 record\_element3 = record\_element\_NO(record\_element3, refine\_element\_NO, N\_e)  
 record\_element4 = record\_element\_NO(record\_element4, refine\_element\_NO, N\_e)  
 record\_element5 = record\_element\_NO(record\_element5, refine\_element\_NO, N\_e)  
 record\_element6 = record\_element\_NO(record\_element6, refine\_element\_NO, N\_e)  
  
 # analysis shear-force & moment & stress of special element  
 analysis\_element = record\_element6  
 count = 0  
 for elemnet\_i in analysis\_element:  
 element\_no = elemnet\_i  
 if count == 0:  
 xs, stress\_xdir\_cpos, stress\_xdir\_cneg = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1],  
 vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1],  
 plot\_picture=False)  
 else:  
 xs\_ = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1], vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1], plot\_picture=False)[0]  
 xs\_ += xs[-1]  
 stress\_xdir\_cpos\_, stress\_xdir\_cneg\_ = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1],  
 vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1],  
 plot\_picture=False)[-2:]  
 stress\_xdir\_cpos = np.hstack([stress\_xdir\_cpos, stress\_xdir\_cpos\_])  
 stress\_xdir\_cneg = np.hstack([stress\_xdir\_cneg, stress\_xdir\_cneg\_])  
 xs = np.hstack([xs, xs\_])  
 # find max axial load  
 if np.max(np.fabs(stress\_xdir\_cpos)) > np.max(np.fabs(stress\_xdir\_cneg)):  
 stress\_xdir = stress\_xdir\_cpos  
 else:  
 stress\_xdir = stress\_xdir\_cneg  
  
 count += 1  
 plt.plot(xs, stress\_xdir, c='r')  
 plt.show()  
  
 # show origin structure  
 show\_picture(ele\_coor, 'k')  
  
 # show deformation structure  
 ele\_u\_mul = ele\_u \* u\_mul  
 for e\_i in range(N\_e):  
 ele\_i\_ori = ele\_coor[e\_i] + np.array(  
 [ele\_u\_mul[e\_i][0], ele\_u\_mul[e\_i][1], ele\_u\_mul[e\_i][3], ele\_u\_mul[e\_i][4]])  
 ele\_i\_L = vecL[e\_i]  
 ele\_i\_the = vecTheta[e\_i]  
 xs, ys = beam\_scatter(ele\_i\_ori, ele\_i\_L, ele\_i\_the,  
 ele\_u\_mul[e\_i])  
 plt.scatter(xs, ys, c='r', s=10)  
 plt.scatter(xs[0], ys[0], c='b', s=20)  
 plt.scatter(xs[-1], ys[-1], c='b', s=20)  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main2()

change\_d\_main.py

import numpy as np  
from element import get\_vecA, get\_vecL, get\_vecE, get\_vecTheta, get\_density, get\_ele\_coor\_Q5  
from element import get\_vecL\_from\_coor, get\_vecTheta\_from\_coor  
from node import node\_member  
from connectivity\_matrix import get\_mtxEFT  
from global\_stiff\_matrix import get\_mtx\_K\_glo  
from add\_BC import mtx\_K\_glo\_add\_BC  
from add\_BC import vecf\_add\_BC  
from show\_picture import show\_picture  
from show\_picture import beam\_scatter  
from distribute\_force import self\_weight\_ele\_to\_node  
from distribute\_force import add\_q\_force\_ele\_to\_node  
from analy\_shearMoment import analy\_shearMoment  
from refine\_mesh\_para import refine\_mesh\_para  
from refine\_mesh\_para import refine\_mesh\_coor  
from refine\_mesh\_para import record\_element\_NO  
import matplotlib.pyplot as plt  
  
# exaggerating deformation  
u\_mul = 1000  
  
  
def main2(distance):  
 # get element and node data  
 ele\_coor = get\_ele\_coor\_Q5(distance)  
 vecL = get\_vecL\_from\_coor(ele\_coor)  
 vecTheta = get\_vecTheta\_from\_coor(ele\_coor)  
 vecA = get\_vecA()  
 vecE = get\_vecE()  
 density = get\_density()  
 N\_gdof, vecFix, vecF = node\_member()  
 mtxEFT = get\_mtxEFT()  
  
 # record element to analysis  
 record\_element1 = [1]  
 record\_element2 = [2]  
 record\_element3 = [3]  
 record\_element4 = [4]  
 record\_element5 = [5]  
 record\_element6 = [6]  
  
 # refine element  
  
 element\_number = 6  
 refine\_element\_list = np.arange(1, element\_number + 1)  
 refine\_times = 1  
 for i in range(1,refine\_times+1):  
 element\_number = element\_number\*2  
 refine\_element\_list = np.hstack([refine\_element\_list, np.arange(1, element\_number+1)])  
  
 for refine\_element\_NO in refine\_element\_list:  
 mtxEFT, vecE, vecA, vecL, vecTheta = refine\_mesh\_para(mtxEFT, vecE, vecA, vecL, vecTheta, refine\_element\_NO)  
 ele\_coor = refine\_mesh\_coor(ele\_coor, refine\_element\_NO)  
 # adjust F distribute  
 # Zeroing force and add force  
 vecF = np.zeros([np.max(mtxEFT)])  
 vecF[17 - 1] += -5000  
 q\_force\_global = np.array([-1000.0, 0, 0])  
 # record\_q\_force should exert on which elements  
 record\_q\_force\_element3 = np.array([3])  
 if refine\_element\_NO in record\_q\_force\_element3:  
 record\_q\_force\_element3 = np.hstack([record\_q\_force\_element3, refine\_element\_NO])  
 for q\_force\_element\_i in record\_q\_force\_element3:  
 vecF += add\_q\_force\_ele\_to\_node(mtxEFT, q\_force\_element\_i, q\_force\_global, vecL, vecTheta)  
  
 # add self\_weight  
 N\_e = mtxEFT.shape[0]  
 for N\_e\_i in range(N\_e):  
 vecF\_weight = self\_weight\_ele\_to\_node(mtxEFT, N\_e\_i + 1, vecA, vecL, vecTheta, density)  
 vecF += vecF\_weight  
  
 # create stiff matrix  
 mtx\_K\_glo = get\_mtx\_K\_glo(mtxEFT, np.max(mtxEFT), N\_e, vecE, vecA, vecL, vecTheta)  
  
 # add boundary condition  
 mtx\_K\_glo\_added\_BC = mtx\_K\_glo\_add\_BC(mtx\_K\_glo, vecFix)  
 vecF\_added\_bc = vecf\_add\_BC(vecF, vecFix)  
  
 # calculate vecU  
 inv\_k = np.linalg.inv(mtx\_K\_glo\_added\_BC)  
 vecU = inv\_k @ vecF\_added\_bc  
  
 # post process calculate external-force  
 F\_ext = mtx\_K\_glo @ vecU  
  
 # get external-force exert on element  
 # get element deformation  
 ele\_f\_ext = np.zeros([N\_e, 6])  
 ele\_u = np.zeros([N\_e, 6])  
 ele\_u\_no = mtxEFT  
 for i in range(N\_e):  
 for j in range(6):  
 ele\_u[i][j] = vecU[ele\_u\_no[i][j] - 1]  
 ele\_f\_ext[i][j] = F\_ext[ele\_u\_no[i][j] - 1]  
  
 # record element to analysis  
 record\_element1 = record\_element\_NO(record\_element1, refine\_element\_NO, N\_e)  
 record\_element2 = record\_element\_NO(record\_element2, refine\_element\_NO, N\_e)  
 record\_element3 = record\_element\_NO(record\_element3, refine\_element\_NO, N\_e)  
 record\_element4 = record\_element\_NO(record\_element4, refine\_element\_NO, N\_e)  
 record\_element5 = record\_element\_NO(record\_element5, refine\_element\_NO, N\_e)  
 record\_element6 = record\_element\_NO(record\_element6, refine\_element\_NO, N\_e)  
  
 # analysis shear-force & moment & stress of special element  
 analysis\_element = record\_element6  
 count = 0  
 for elemnet\_i in analysis\_element:  
 element\_no = elemnet\_i  
 if count == 0:  
 xs, stress\_xdir\_cpos, stress\_xdir\_cneg = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1],  
 vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1],  
 plot\_picture=False)  
 else:  
 xs\_ = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1], vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1], plot\_picture=False)[0]  
 xs\_ += xs[-1]  
 stress\_xdir\_cpos\_, stress\_xdir\_cneg\_ = analy\_shearMoment(ele\_u[element\_no - 1], vecTheta[element\_no - 1],  
 vecE[element\_no - 1],  
 vecA[element\_no - 1], vecL[element\_no - 1],  
 plot\_picture=False)[-2:]  
 stress\_xdir\_cpos = np.hstack([stress\_xdir\_cpos, stress\_xdir\_cpos\_])  
 stress\_xdir\_cneg = np.hstack([stress\_xdir\_cneg, stress\_xdir\_cneg\_])  
 xs = np.hstack([xs, xs\_])  
 # find max axial load  
 if np.max(np.fabs(stress\_xdir\_cpos)) > np.max(np.fabs(stress\_xdir\_cneg)):  
 stress\_xdir = stress\_xdir\_cpos  
 else:  
 stress\_xdir = stress\_xdir\_cneg  
  
 count += 1  
 # plt.plot(xs, stress\_xdir, c='r')  
 # plt.show()  
  
 # show origin structure  
 # show\_picture(ele\_coor, 'k')  
  
 # show deformation structure  
 # ele\_u\_mul = ele\_u \* u\_mul  
 # for e\_i in range(N\_e):  
 # ele\_i\_ori = ele\_coor[e\_i] + np.array(  
 # [ele\_u\_mul[e\_i][0], ele\_u\_mul[e\_i][1], ele\_u\_mul[e\_i][3], ele\_u\_mul[e\_i][4]])  
 # ele\_i\_L = vecL[e\_i]  
 # ele\_i\_the = vecTheta[e\_i]  
 # xs, ys = beam\_scatter(ele\_i\_ori, ele\_i\_L, ele\_i\_the,  
 # ele\_u\_mul[e\_i])  
 # plt.scatter(xs, ys, c='r', s=10)  
 # plt.scatter(xs[0], ys[0], c='b', s=20)  
 # plt.scatter(xs[-1], ys[-1], c='b', s=20)  
 # plt.show()  
 return np.max(np.fabs(stress\_xdir))  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 distance=np.linspace(0,20,500)  
 for i in distance:  
 stress\_max=main2(i)  
 if i ==0:  
 stress\_maxs=np.array([stress\_max])  
 else:  
 stress\_maxs=np.hstack([stress\_maxs,stress\_max])  
 np.save('stress\_maxs',stress\_maxs)

connectivity\_matrix.py

import numpy as np  
  
  
# connect global\_vecU to element, example[x\_dir1 y\_dir1 theta1 x\_dir2 y\_dir2 theta2]  
def get\_mtxEFT():  
 mtxEFT = np.array([[13, 14, 15, 16, 17, 18],  
 [10, 11, 12, 13, 14, 15],  
 [7, 8, 9, 16, 17, 18],  
 [10, 11, 12, 7, 8, 9],  
 [1, 2, 3, 10, 11, 12],  
 [4, 5, 6, 7, 8, 9]  
 ])  
 return mtxEFT  
  
# connect node to vecU  
def get\_nodeANDdof\_table():  
 nodeANDdof = np.array([  
 [1,2,3],  
 [4,5,6],  
 [7,8,9],  
 [10,11,12],  
 [13,14,15],  
 [16,17,18]  
 ])  
 return nodeANDdof

element.py

import numpy as np  
from math import atan2,pi  
  
'''store element parameters  
'''  
  
  
# element length  
def get\_vecL():  
 vecL = np.array([3, 2, 2, 3, 2, 2.385])  
 return vecL  
  
  
# element area  
def get\_vecA():  
 vecA = np.array(  
 [62500e-6, 62500e-6, 62500e-6, 62500e-6, 62500e-6, 62500e-6])  
 return vecA  
  
  
# element young's module  
def get\_vecE():  
 vecE = np.array([210e9, 210e9, 210e9, 210e9, 210e9, 210e9])  
 return vecE  
  
  
# element theta of global coordinate to local coordinate (degree)  
def get\_vecTheta():  
 vecTheta = np.array([0, 90, 90, 0, 90, 123.0239])  
 return vecTheta  
  
  
# element density  
def get\_density():  
 return 7900  
  
  
# element global coordinate  
def get\_ele\_coor():  
 ele\_coor = np.array([  
 [0, 4, 3, 4],  
 [0, 2, 0, 4],  
 [3, 2, 3, 4],  
 [0, 2, 3, 2],  
 [0, 0, 0, 2],  
 [4.3, 0, 3, 2]  
 ])  
 return ele\_coor  
  
# element global coordinate for Q5  
def get\_ele\_coor\_Q5(x):  
 ele\_coor = np.array([  
 [0, 4, 3, 4],  
 [0, 2, 0, 4],  
 [3, 2, 3, 4],  
 [0, 2, 3, 2],  
 [0, 0, 0, 2],  
 [x, 0, 3, 2]  
 ])  
 return ele\_coor  
  
# element length  
def get\_vecL\_from\_coor(ele\_coor):  
 count = 0  
 for ele in ele\_coor:  
 x1 = ele[0]  
 y1 = ele[1]  
 x2 = ele[2]  
 y2 = ele[3]  
 L = ((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2) \*\* 0.5  
 if count == 0:  
 vecL = np.array([L])  
 else:  
 vecL = np.hstack([vecL, L])  
 count += 1  
 return vecL  
  
# element theta of global coordinate to local coordinate (degree)  
def get\_vecTheta\_from\_coor(ele\_coor):  
 count = 0  
 for ele in ele\_coor:  
 x1 = ele[0]  
 y1 = ele[1]  
 x2 = ele[2]  
 y2 = ele[3]  
 Theta = atan2((y2 - y1), (x2 - x1))  
 if count == 0:  
 vecTheta = np.array([Theta])  
 else:  
 vecTheta = np.hstack([vecTheta, Theta])  
 count += 1  
 return vecTheta/pi\*180.0

distribute\_force.py

import math  
import numpy as np  
  
  
def add\_q\_force(L, qx\_global, theta):  
 rad = theta \* 3.1415926 / 180.0  
 c = math.cos(rad)  
 s = math.sin(rad)  
 T = np.array([  
 [c, -s, 0],  
 [s, c, 0],  
 [0, 0, 1]  
 ])  
 T\_t = np.array([  
 [c, s, 0],  
 [-s, c, 0],  
 [0, 0, 1]  
 ])  
 qx\_local = T\_t @ qx\_global  
 vecf\_local = L / 2 \* np.array([  
 [qx\_local[0]],  
 [qx\_local[1]],  
 [qx\_local[1] \* L / 6],  
 [qx\_local[0]],  
 [qx\_local[1]],  
 [-qx\_local[1] \* L / 6],  
 ])  
 T2 = np.kron(np.eye(2, dtype=float), T)  
 vecf\_global = T2 @ vecf\_local  
 return vecf\_global  
  
  
def self\_weight\_ele\_to\_node(mtxEFT, e\_i, vecA, vecL, vectheta, density):  
 vecf\_ = np.zeros([np.max(mtxEFT)])  
 node\_list = mtxEFT[e\_i - 1]  
 qx\_global = np.array([0, -density \* vecA[e\_i - 1] \* 9.81, 0])  
 self\_weight\_vecf = add\_q\_force(vecL[e\_i - 1], qx\_global, vectheta[e\_i - 1])  
 count = 0  
 for node\_i in node\_list:  
 vecf\_[node\_i - 1] = self\_weight\_vecf[count]  
 count += 1  
 return vecf\_  
  
def add\_q\_force\_ele\_to\_node(mtxEFT, e\_i ,qx\_global,vecL, vectheta):  
 vecf\_ = np.zeros([np.max(mtxEFT)])  
 node\_list = mtxEFT[e\_i - 1]  
 self\_weight\_vecf = add\_q\_force(vecL[e\_i - 1], qx\_global, vectheta[e\_i - 1])  
 count = 0  
 for node\_i in node\_list:  
 vecf\_[node\_i - 1] = self\_weight\_vecf[count]  
 count += 1  
 return vecf\_  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 pass

global\_stiff\_matrix.py

import numpy as np  
import math  
  
  
def main():  
 pass  
  
  
def get\_mtx\_K\_glo(mtxFET, N\_gdof, N\_e, vecE, vecA, vecL, vecTheta):  
 mtx\_K\_glo = np.zeros([N\_gdof, N\_gdof])  
 for e\_i in range(N\_e):  
 mtx\_K\_e = get\_mtx\_K\_e(vecE[e\_i], vecA[e\_i], vecL[e\_i], vecTheta[e\_i])  
 for i in range(6):  
 for j in range(6):  
 mtx\_K\_glo[mtxFET[e\_i, i] - 1, mtxFET[e\_i, j] - 1] = mtx\_K\_glo[mtxFET[e\_i, i] - 1, mtxFET[e\_i, j] - 1] + \  
 mtx\_K\_e[i, j]  
  
 return mtx\_K\_glo  
  
  
def get\_mtx\_K\_e(E, A, L, D):  
 R = D \* 3.1415926 / 180.0  
 c = math.cos(R)  
 s = math.sin(R)  
 I = (1 / 12.0) \* A \* A  
 T = np.array([  
 [c, s, 0, 0, 0, 0],  
 [-s, c, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0],  
 [0, 0, 0, c, s, 0],  
 [0, 0, 0, -s, c, 0],  
 [0, 0, 0, 0, 0, 1]  
 ])  
 T\_t = np.array([  
 [c, -s, 0, 0, 0, 0],  
 [s, c, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0],  
 [0, 0, 0, c, -s, 0],  
 [0, 0, 0, s, c, 0],  
 [0, 0, 0, 0, 0, 1]  
 ])  
 K\_bar = E \* A / L \* np.array([  
 [1, -1],  
 [-1, 1]  
 ])  
 K\_beam = E \* I / (L \*\* 3) \* np.array([  
 [12, 6 \* L, -12, 6 \* L],  
 [6 \* L, 4 \* L \*\* 2, -6 \* L, 2 \* L \*\* 2],  
 [-12, -6 \* L, 12, -6 \* L],  
 [6 \* L, 2 \* L \*\* 2, -6 \* L, 4 \* L \*\* 2]  
 ])  
  
 K = np.array([  
 [K\_bar[0][0], 0, 0, K\_bar[0][1], 0, 0],  
 [0, K\_beam[0][0], K\_beam[0][1], 0, K\_beam[0][2], K\_beam[0][3]],  
 [0, K\_beam[1][0], K\_beam[1][1], 0, K\_beam[1][2], K\_beam[1][3]],  
 [K\_bar[1][0], 0, 0, K\_bar[1][1], 0, 0],  
 [0, K\_beam[2][0], K\_beam[2][1], 0, K\_beam[2][2], K\_beam[2][3]],  
 [0, K\_beam[3][0], K\_beam[3][1], 0, K\_beam[3][2], K\_beam[3][3]],  
 ])  
 mtx\_K\_e = T\_t @ K @ T  
  
 return mtx\_K\_e  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

node.py

import numpy as np  
  
'''store node parameters'''  
  
  
def node\_member():  
 N\_gdof = 18  
 vecFix = np.array([1, 2, 3, 5])  
 vecF = np.zeros([N\_gdof])  
  
 return N\_gdof, vecFix, vecF  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 node\_member()

refine\_mesh\_para.py

import numpy as np  
  
  
def refine\_mesh\_para(mtxEFT, vecE, vecA, vecL, vecTheta, e\_i):  
 # change mtxEFT  
 mtxEFT = np.copy(mtxEFT)  
 node\_max = np.max(mtxEFT)  
 element\_node = mtxEFT[e\_i - 1]  
 node1 = element\_node[:3]  
 node2 = element\_node[-3:]  
 add\_node = np.array([node\_max + 1, node\_max + 2, node\_max + 3])  
 element1 = np.hstack([node1, add\_node])  
 element2 = np.hstack([add\_node, node2])  
 # add element in EFT  
 mtxEFT[e\_i - 1] = element1  
 mtxEFT = np.vstack([mtxEFT, element2])  
  
 # change element\_para  
 vecE = np.copy(vecE)  
 vecA = np.copy(vecA)  
 vecL = np.copy(vecL)  
 vecTheta = np.copy(vecTheta)  
  
 vecL[e\_i - 1] = vecL[e\_i - 1] / 2.0  
  
 vecE = np.hstack([vecE, vecE[e\_i - 1]])  
 vecA = np.hstack([vecA, vecA[e\_i - 1]])  
 vecL = np.hstack([vecL, vecL[e\_i - 1]])  
 vecTheta = np.hstack([vecTheta, vecTheta[e\_i - 1]])  
  
 return mtxEFT, vecE, vecA, vecL, vecTheta  
 pass  
  
  
def refine\_mesh\_coor(ele\_coor, ei):  
 ele\_coor = np.copy(ele\_coor)  
 node1\_coordinate = ele\_coor[ei - 1][:2]  
 node2\_coordinate = ele\_coor[ei - 1][-2:]  
 node3\_coordinate = 0.5 \* (node1\_coordinate + node2\_coordinate)  
  
 ele1 = np.hstack([node1\_coordinate, node3\_coordinate])  
 ele2 = np.hstack([node3\_coordinate, node2\_coordinate])  
  
 ele\_coor[ei - 1] = ele1  
 ele\_coor = np.vstack([ele\_coor, ele2])  
  
 return ele\_coor  
  
  
# record element split number  
def record\_element\_NO(record\_elements, refine\_element\_NO,N\_e):  
 if refine\_element\_NO in record\_elements:  
 index=np.where(record\_elements==refine\_element\_NO)[0][0]  
  
 record\_elements.insert(index+1,N\_e)  
  
 return record\_elements  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import connectivity\_matrix  
  
 vecL = np.array([3, 2, 2, 3, 2, 2.385])  
 vecA = np.array(  
 [62500e-6, 62500e-6, 62500e-6, 62500e-6, 62500e-6, 62500e-6])  
 vecE = np.array([210e9, 210e9, 210e9, 210e9, 210e9, 210e9])  
 vecTheta = np.array([0, 90, 90, 0, 90, 123.0239])  
  
 mtxEFT = connectivity\_matrix.get\_mtxEFT()  
 mtxEFT, vecE, vecA, vecL, vecTheta = refine\_mesh\_para(mtxEFT, vecE, vecA, vecL, vecTheta, 1)  
 from global\_stiff\_matrix import get\_mtx\_K\_glo  
  
 mtx\_K\_glo = get\_mtx\_K\_glo(mtxEFT, np.max(mtxEFT), mtxEFT.shape[0], vecE, vecA, vecL, vecTheta)  
 pass

show\_picture.py

import numpy as np  
import math  
import matplotlib.pyplot as plt  
  
  
def beam\_scatter(ele\_ori, ele\_L, ele\_the, ele\_u):  
 R = ele\_the \* 3.1415926 / 180.0  
 c = math.cos(R)  
 s = math.sin(R)  
 T = np.array([  
 [c, s, 0, 0, 0, 0],  
 [-s, c, 0, 0, 0, 0],  
 [0, 0, 1, 0, 0, 0],  
 [0, 0, 0, c, s, 0],  
 [0, 0, 0, -s, c, 0],  
 [0, 0, 0, 0, 0, 1]  
 ])  
 ele\_u\_e = T @ ele\_u  
 v1 = ele\_u\_e[1]  
 the1 = ele\_u\_e[2]  
 v2 = ele\_u\_e[4]  
 the2 = ele\_u\_e[5]  
  
 def N1(x):  
 return (1 / ele\_L \*\* 3) \* (ele\_L - x) \*\* 2 \* (2 \* x + ele\_L)  
  
 def N2(x):  
 return (1 / ele\_L \*\* 2) \* (ele\_L - x) \*\* 2 \* x  
  
 def N3(x):  
 return (1 / ele\_L \*\* 3) \* (3 \* ele\_L - 2 \* x) \* x \*\* 2  
  
 def N4(x):  
 return (1 / ele\_L \*\* 2) \* (x - ele\_L) \* x \*\* 2  
  
 xs = np.linspace(0, ele\_L, 100)  
 vs = N1(xs) \* v1 + N2(xs) \* the1 + N3(xs) \* v2 + N4(xs) \* the2  
  
 ele\_the\_rad = ele\_the / 180.0 \* 3.1415926  
  
 vs = vs - vs[0]  
  
 xs\_ = xs \* math.cos(ele\_the\_rad) - vs \* math.sin(ele\_the\_rad)  
 vs\_ = xs \* math.sin(ele\_the\_rad) + vs \* math.cos(ele\_the\_rad)  
  
 xs\_glo = xs\_ + ele\_ori[0]  
 vs\_glo = vs\_ + ele\_ori[1]  
  
 return xs\_glo, vs\_glo  
  
  
def make\_scatter\_point(p1, p2, n=50):  
 x = np.linspace(p1[0], p2[0], n)  
 y = np.linspace(p1[1], p2[1], n)  
 return x, y  
  
  
def show\_picture(matrix, color):  
 ele\_N = matrix.shape[0]  
 for i in range(ele\_N):  
 p1x = matrix[i][0]  
 p1y = matrix[i][1]  
 p2x = matrix[i][2]  
 p2y = matrix[i][3]  
 xs, ys = make\_scatter\_point([p1x, p1y], [p2x, p2y])  
 plt.scatter(xs, ys, c=color, s=2)