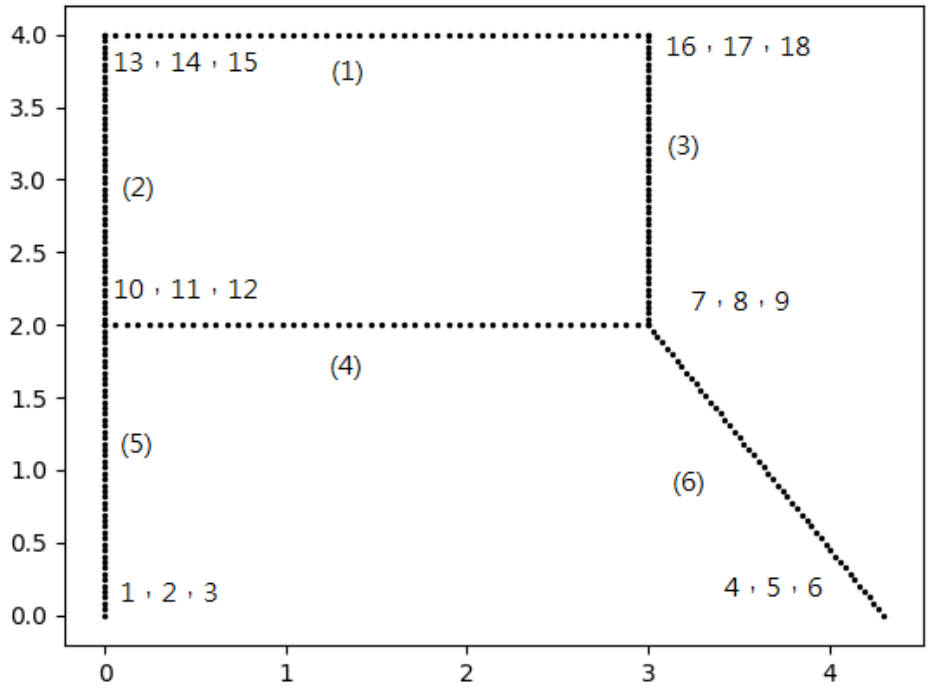
有限元素 project2

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

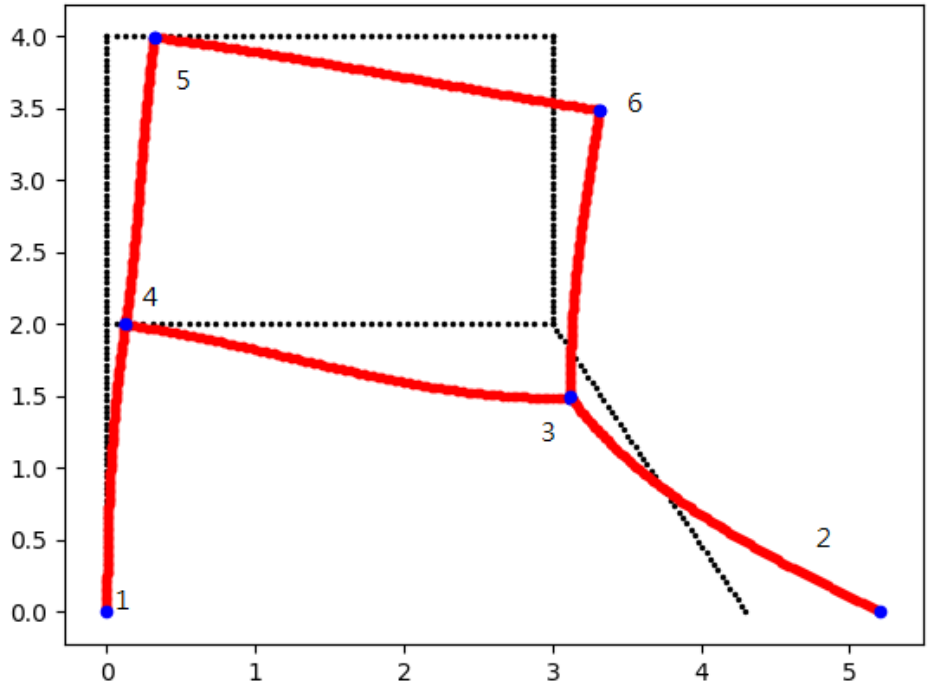
1.02e+08	1.73e+02	-1.02e+08	-1.02e+08	-1.73e+02	-1.02e+08
1.73e+02	6.56e+09	2.74e+00	-1.73e+02	-6.56e+09	2.74e+00
-1.02e+08	2.74e+00	1.36e+08	1.02e+08	-2.74e+00	6.83e+07
-1.02e+08	-1.73e+02	1.02e+08	1.02e+08	1.73e+02	1.02e+08
-1.73e+02	-6.56e+09	-2.74e+00	1.73e+02	6.56e+09	-2.74e+00
-1.02e+08	2.74e+00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

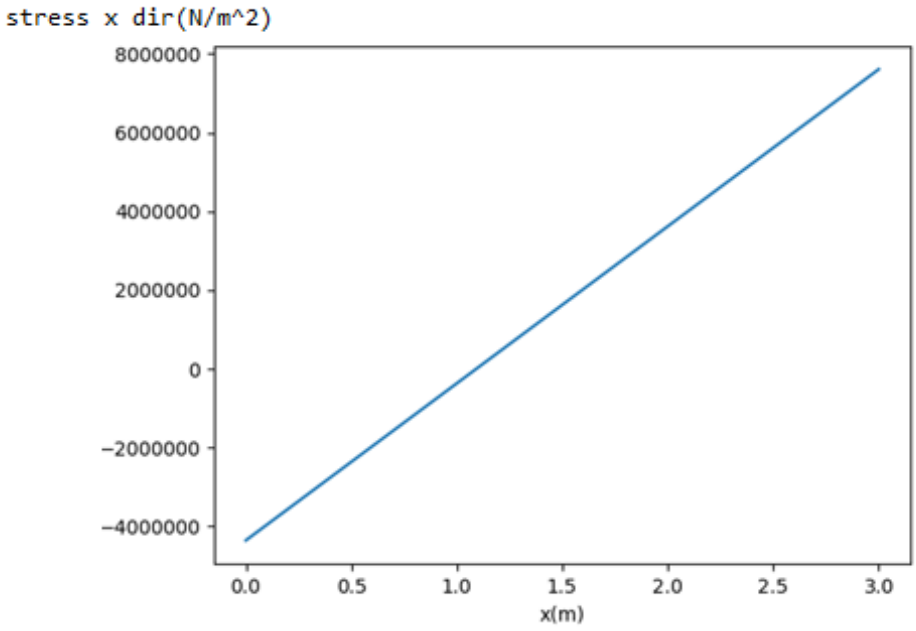


node	Deflection x	Deflection y	Deflection θ
1	0.000e+00	0.000e+00	0.000e+00
2	9.068e-04	0.000e+00	5.830e-04
3	1.241e-04	-5.127e-04	3.256e-05
4	1.228e-04	-6.354e-06	-1.423e-04
5	3.184e-04	-8.543e-06	-1.312e-04
6	3.166e-04	-5.150e-04	-1.569e-04

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。

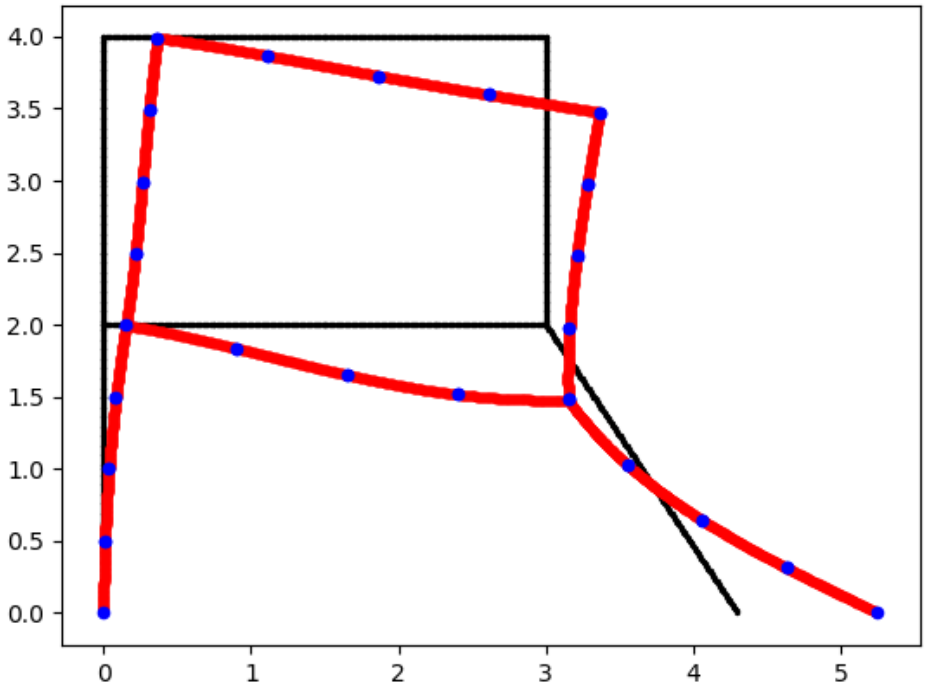


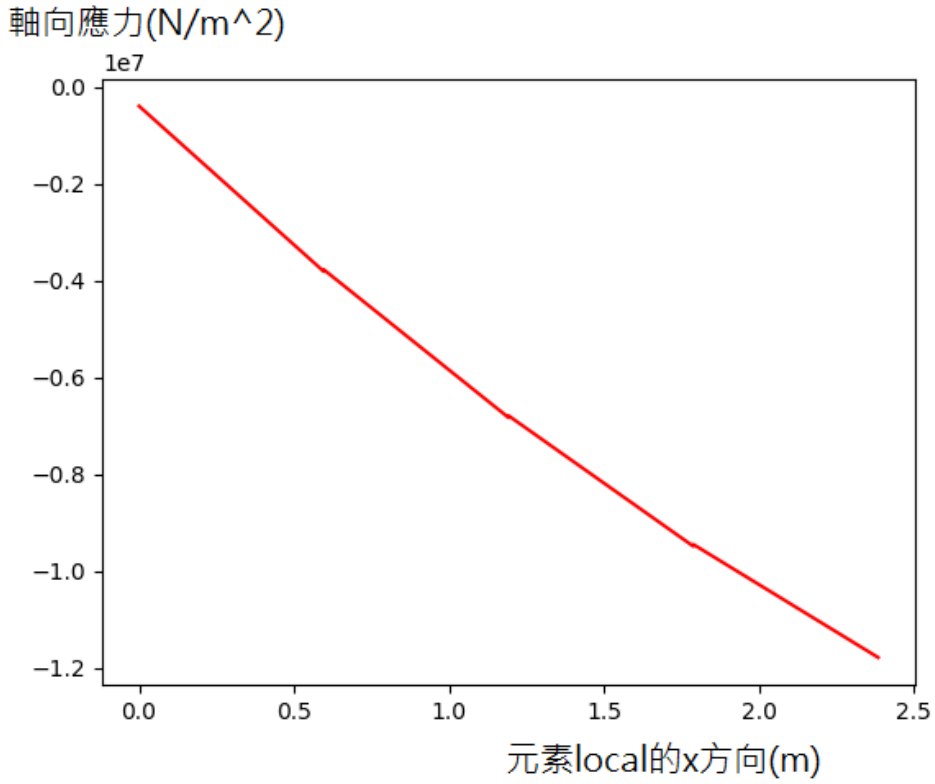
軸向應力最大值：7607603(N/m^2)

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

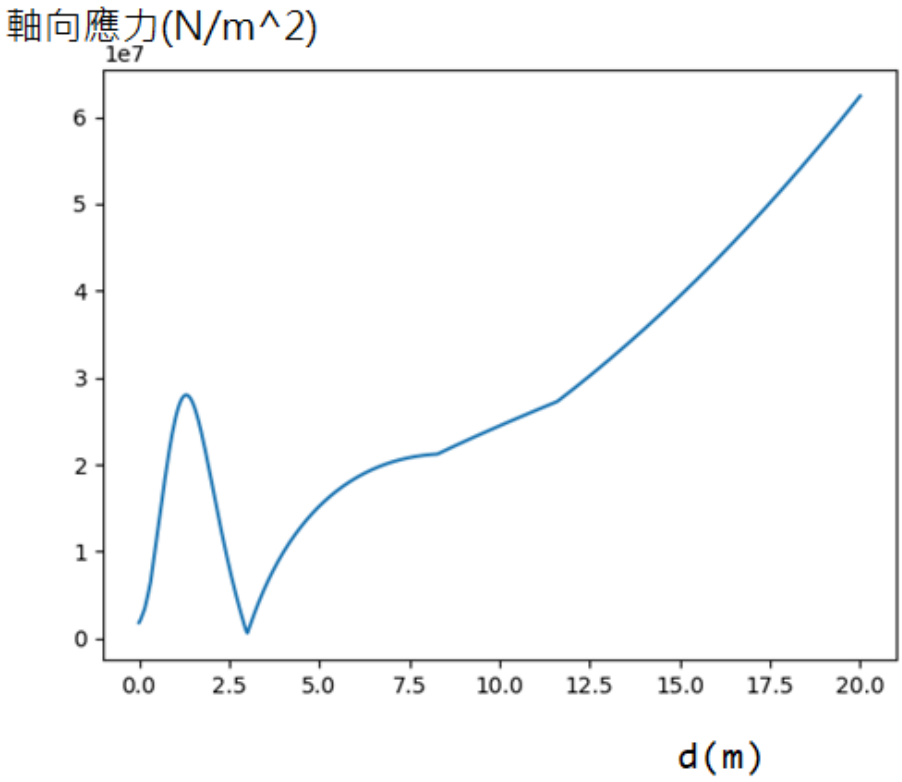
在此題中我把每個元素再細切成四段。

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)。





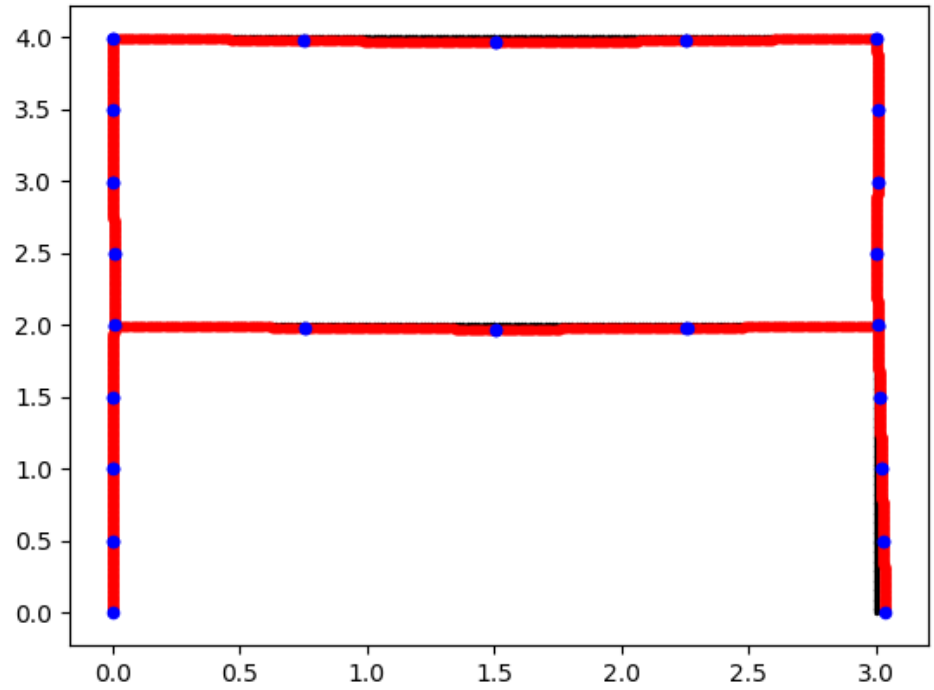
我將距離 d 從 0 到 20 以等間距 500 次分析，計算最大軸向應力的值如下圖



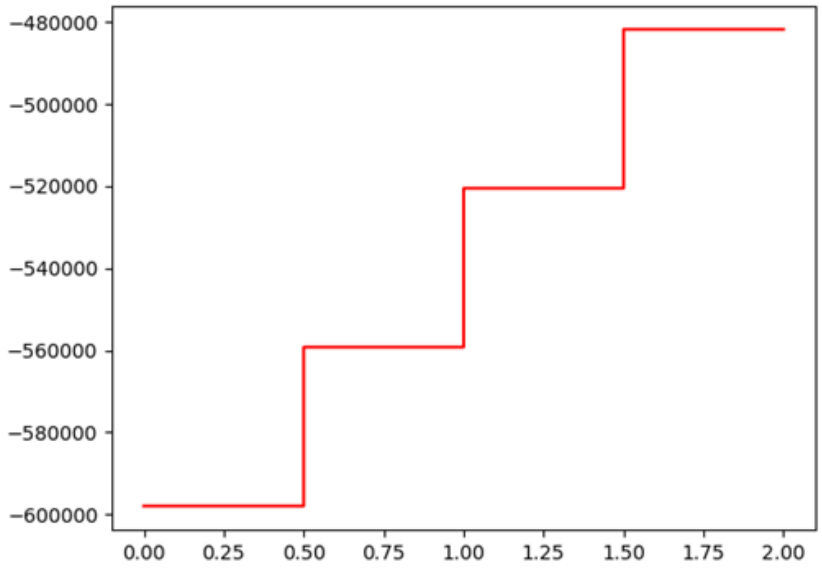
最小軸向應力值為 618566N/m^2

此時 d=3m

結構如下圖。



軸向應力(N/m^2)



元素local的x方向(m)

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)
```