结构矩阵分析-期中大作业

谭泽人

May 2019

1 程序说明

1.1 概述

本程序采用 Python 3 编写，其中用到了 Python 的 `Numpy` 和 `Scipy` 库。程序可用于静力分析，利用矩阵位移法求解静定和超静定结构的内力和位移。输入文件的名称为 SM90.IPT，输出文件的名称为 SMCAI90.out，代码采用 Python3.6 编译。

1.2 输入文件的格式

输入文件的第一行为 1，表示问题为静力分析；第二行为 5 个整数：NElem, NJoint, NGlbDOF, NJLoad, NELoad，分别代表单元数，节点数，整体自由度数，节点荷载数，单元荷载数。

之后的 NJoint 行，每行两个实数（节点的坐标）和三个整数（节点的位移编码）。

之后的 NElem 行，每行两个整数（两个节点的编号）和两个实数（抗拉刚度和抗弯刚度）。

接下来的 NJLoad 行，每行两个整数（节点的编号和力的方向）和一个实数（力的大小）。

接下来的 NELoad，每行两个整数（单元的编号和力的方向）和两个实数（力的作用位置和里的大小）。

1.3 输出文件的格式

第一行为 10，为固定的输出。

接下来的 NElem 行，每行 6 个实数，分别表示每个单元的整体位移向量。

接下来的 NElem 行，每行 6 个实数，分别表示每个单元的杆端力向量。

2 源程序清单

这里只列出程序中类和函数的名称，具体实现请见

2.1 类定义

- `JOint`（节点类）
- `Element`（单元类）

- `JOintLoad`（节点荷载类）
- `ElementLoad`（单元荷载类）

2.2 函数定义

- `SetElemProp`（根据输入设置单元的属性）
- `TransMatrix`（求单元转换矩阵）
- `SetMatBand`（变带宽稀疏矩阵存储）
- `varBandSolv`（利用变带宽矩阵求解节点位移）
- `solveDisp`（求解节点位移）
- `GStifMat`（计算整体刚度矩阵）
- `GLoadVec`（计算整体荷载向量）
- `EStifMat`（计算单元刚度矩阵）
- `EFixendF`（计算单元杆端力）
- `ElemDisp`（计算单元位移向量）
- `ElemForce`（计算单元荷载向量）
- `InputData`（读取输入）
- `OUtputResults`（输出结果）

3 输入数据文件

1				
16	17	35	2	9
-5	0	0	0	1
-5	5	2	3	4
0	0	0	0	0
0	4	8	9	10
0	8	5	6	7
0	8	5	6	11
0	12	12	13	14
0	16	15	16	17
0	16	15	16	27
6	0	0	0	31
6	4	18	19	20
6	4	18	19	32
6	8	21	22	23
6	12	24	25	26
6	16	28	29	30
11	0	0	0	0
11	3	33	34	35
1	2	100000	15000	
2	5	1000000	10000	
3	4	100000	15000	
4	6	100000	15000	
6	7	100000	15000	
7	8	100000	15000	
4	11	1000000	10000	
6	13	1000000	10000	
7	14	1000000	10000	
9	15	1000000	10000	
10	11	100000	15000	
11	13	100000	15000	
13	14	100000	15000	
14	15	100000	15000	
12	17	1000000	10000	
16	17	100000	15000	
2	1	10		
8	1	10		
9	2	0.5	-10	

10	2	0.25	-10
10	2	0.75	-10
5	1	1	-8
6	1	1	-8
7	1	1	-12
8	1	1	-12
15	1	1	-10.2
3	3	0	-0.01

4 输出数据文件

10	0	
0.000000000000000E+000	0.000000000000000E+000	-1.483690993528739E-002
6.418489392069365E-002	2.759801715308387E-004	-8.837116481841401E-003
6.418489392069365E-002	2.759801715308387E-004	-8.837116481841401E-003
6.970773226163729E-002	-8.923779372194643E-003	1.658235619974010E-003
0.000000000000000E+000	-1.000000000000000E-002	0.000000000000000E+000
1.790031640466343E-002	-9.682770861283222E-003	-8.381089914460826E-003
1.790031640466343E-002	-9.682770861283222E-003	-8.381089914460826E-003
6.970773226163729E-002	-8.923779372194643E-003	-1.300427413935570E-002
6.970773226163729E-002	-8.923779372194643E-003	-1.300427413935570E-002
0.123778757680594	-8.024209268763212E-003	-9.577186728495116E-003
0.123778757680594	-8.024209268763212E-003	-9.577186728495116E-003
0.166117302781533	-8.100825000392333E-003	-1.037724993749355E-002
1.790031640466343E-002	-9.682770861283222E-003	-8.381089914460826E-003
1.776702835360894E-002	-7.356465927695313E-003	-5.817039852099571E-003
6.970773226163729E-002	-8.923779372194643E-003	-1.300427413935570E-002
6.959710063061728E-002	-1.241624155400855E-002	-1.096296276447603E-002
0.123778757680594	-8.024209268763212E-003	-9.577186728495116E-003
0.123696081219148	-1.451581165743998E-002	-1.022946826030545E-002
0.166117302781533	-8.100825000392333E-003	2.860355379934105E-004
0.166002303492634	-1.523919592581086E-002	-7.516256538696082E-003
0.000000000000000E+000	0.000000000000000E+000	-3.754115706553567E-003
1.776702835360894E-002	-7.356465927695313E-003	-5.817039852099571E-003
1.776702835360894E-002	-7.356465927695313E-003	-5.817039852099571E-003
6.959710063061728E-002	-1.241624155400855E-002	-1.096296276447603E-002
6.959710063061728E-002	-1.241624155400855E-002	-1.096296276447603E-002
0.123696081219148	-1.451581165743998E-002	-1.022946826030545E-002
0.123696081219148	-1.451581165743998E-002	-1.022946826030545E-002
0.166002303492634	-1.523919592581086E-002	-7.516256538696082E-003
1.776702835360894E-002	-7.356465927695313E-003	1.683349811975474E-003
1.851247379931633E-002	-1.636160511184592E-003	-5.614927048594705E-003
0.000000000000000E+000	0.000000000000000E+000	0.000000000000000E+000
1.851247379931633E-002	-1.636160511184592E-003	-5.614927048594705E-003
5.51960343061678	7.19975214413520	0.000000000000000E+000
5.51960343061678	-7.19975214413520	35.9987607206760
0.438619810049204	-6.17373653047965	-35.9987607206760
0.438619810049204	6.17373653047965	1.776356839400250E-015
7.93072846791944	3.20100911927374	37.8311054177756

7.93072846791944	-3.20100911927374	-25.0270689406806
18.9747872272145	25.4156842950211	68.1683094333979
18.9747872272145	-25.4156842950211	33.4944277466863
22.4892525857858	41.0540416091545	47.9231720942485
22.4892525857858	-9.05404160915451	52.2929943423696
-1.91539329072802	22.8334518502053	27.3338074008214
-1.91539329072802	9.16654814979467	-1.225686219186173E-013
-22.2146751757477	11.0440587592949	-43.1412404927173
-22.2146751757477	60.9559412407051	-106.594406951513
-18.4386051700021	-2.00513807204516	-81.4175998409349
-18.4386051700021	74.0051380720452	-146.613228591336
-13.7794102410517	-24.4046458765138	-79.6268017431910
-13.7794102410517	34.4046458765138	-96.8010735158921
-19.1665481497803	1.91539329072806	-3.552713678800501E-015
-19.1665481497803	18.0846067092719	-48.5076402556316
-183.911648192383	-3.86798277289875	1.421085471520200E-014
-183.911648192383	3.86798277289875	-15.4719310915950
-126.494390657831	51.3845635608482	122.066338043108
-126.494390657831	-51.3845635608482	83.4719162002846
-52.4892525857858	32.9459583908456	63.1413123910515
-52.4892525857858	-32.9459583908456	68.6425211723309
-18.0846067092719	19.1665481497981	28.1585523435612
-18.0846067092719	-19.1665481497981	48.5076402556314
-76.6568533836057	11.7226072226999	7.105427357601002E-015
-76.6568533836057	40.2873918159465	-72.8261970212693
-54.5386837061531	67.2672215094952	128.975467507216
-54.5386837061531	-67.2672215094952	72.8261970212692

5 计算简图

6 弯矩，剪力，轴力图

- 位移图

图 1: 结构位移图

- 轴力图

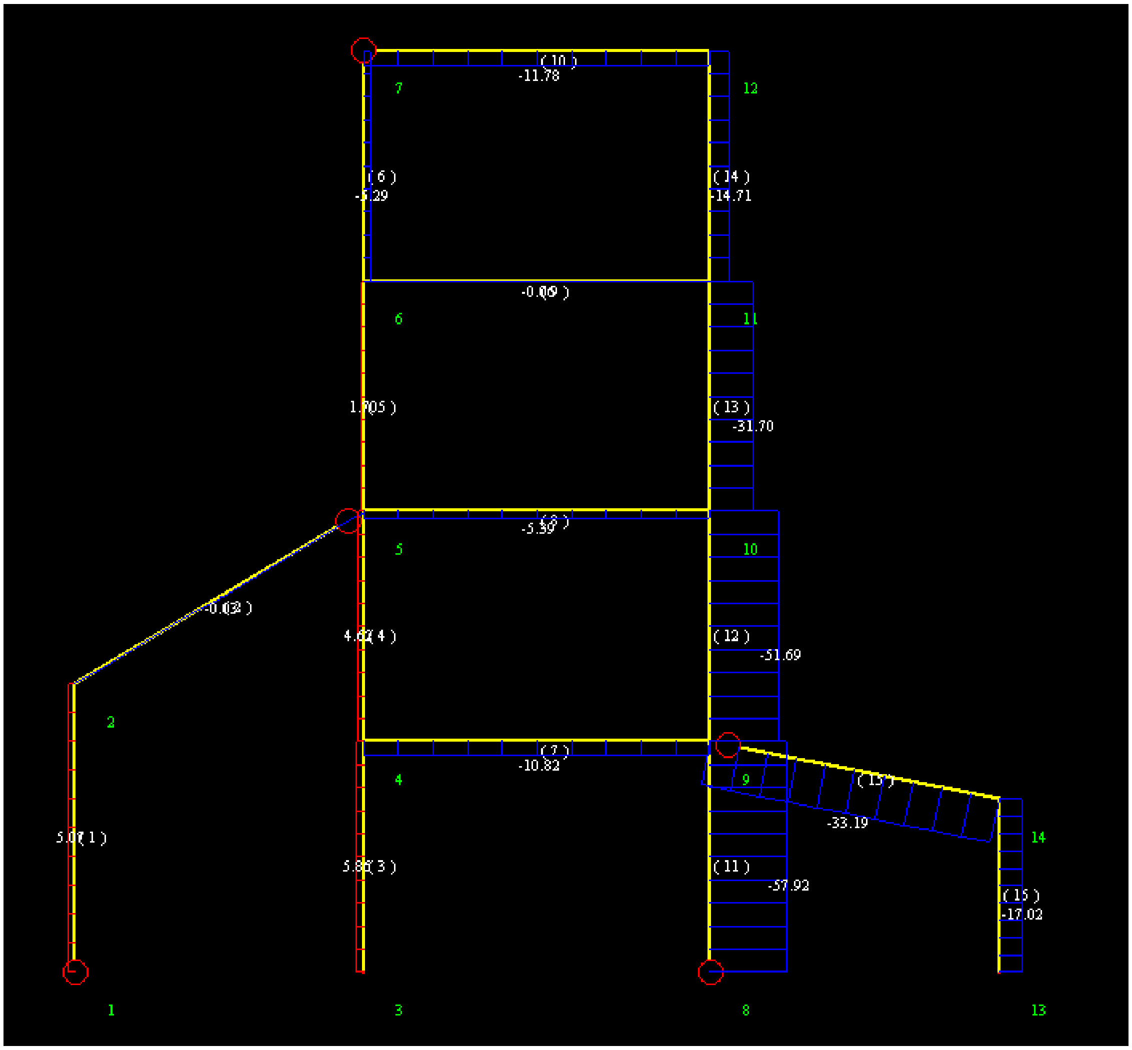
(10)
-11.78
7
12
(6)
(14)
-14.71
6
11
(13)
-31.70
5
10
-5.39
(12)
-51.69
2
(7)
-10.82
4
9
-33.19
14
(11)
-57.92
(15)
-17.02
1
3
8
13

图 2: 轴力图

- 剪力图

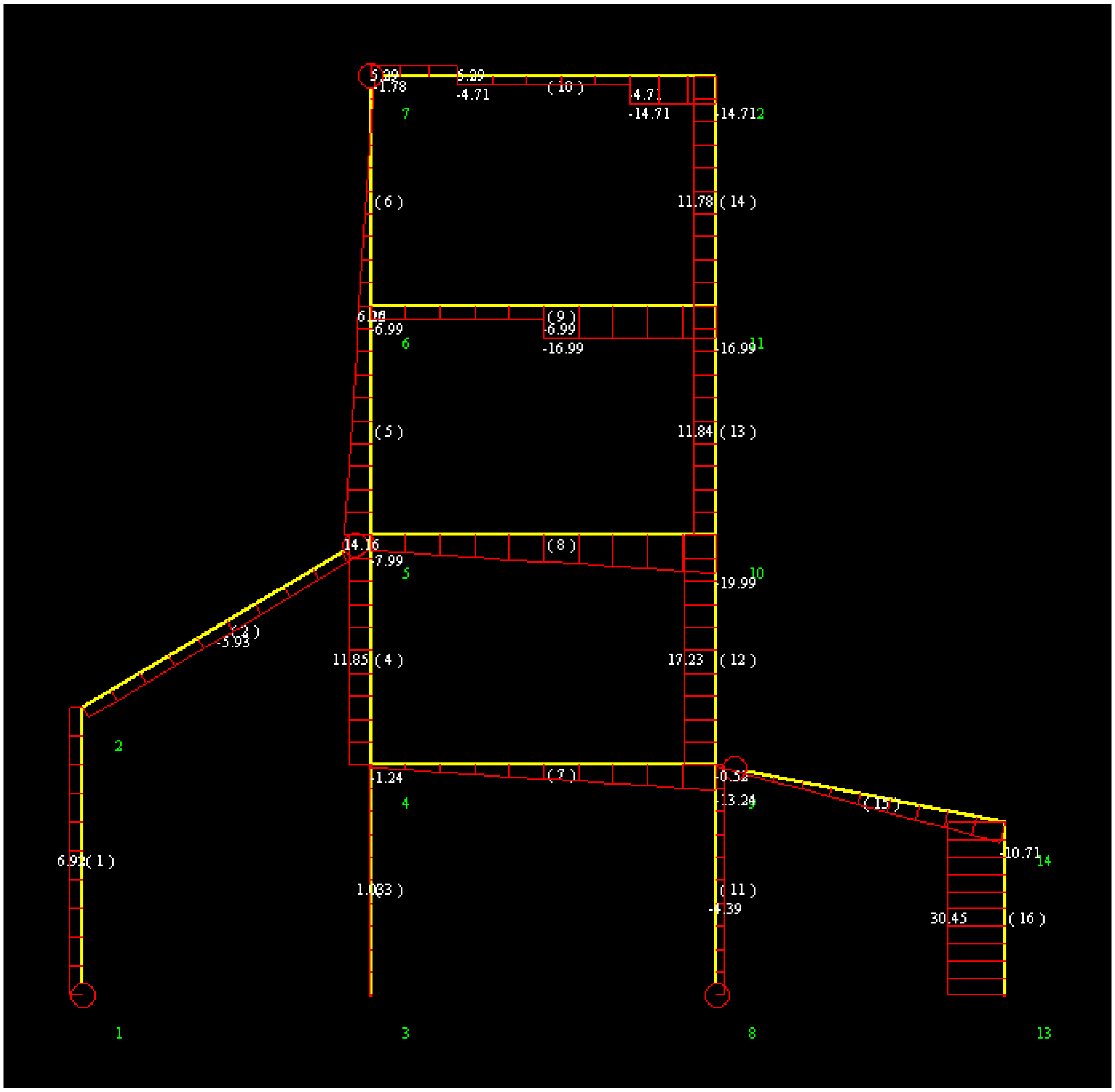
5.29
-1.78
5.29
-4.71
(10)
-4.71
-14.71
-14.71
12
7
(6)
11.78
(14)
6.28
-6.99
(9)
-6.99
-16.99
6
-16.99
11
(5)
11.84
(13)
14.16
-7.99
(8)
5
-19.99
10
(2)
-5.93
11.85
(4)
17.23
(12)
2
-1.24
(7)
-0.52
-13.24
4
(15)
6.93
(1)
-10.71
14
1.033
(11)
-4.39
30.45
(16)
1
3
8
13

图 3: 剪力图

- 弯矩图

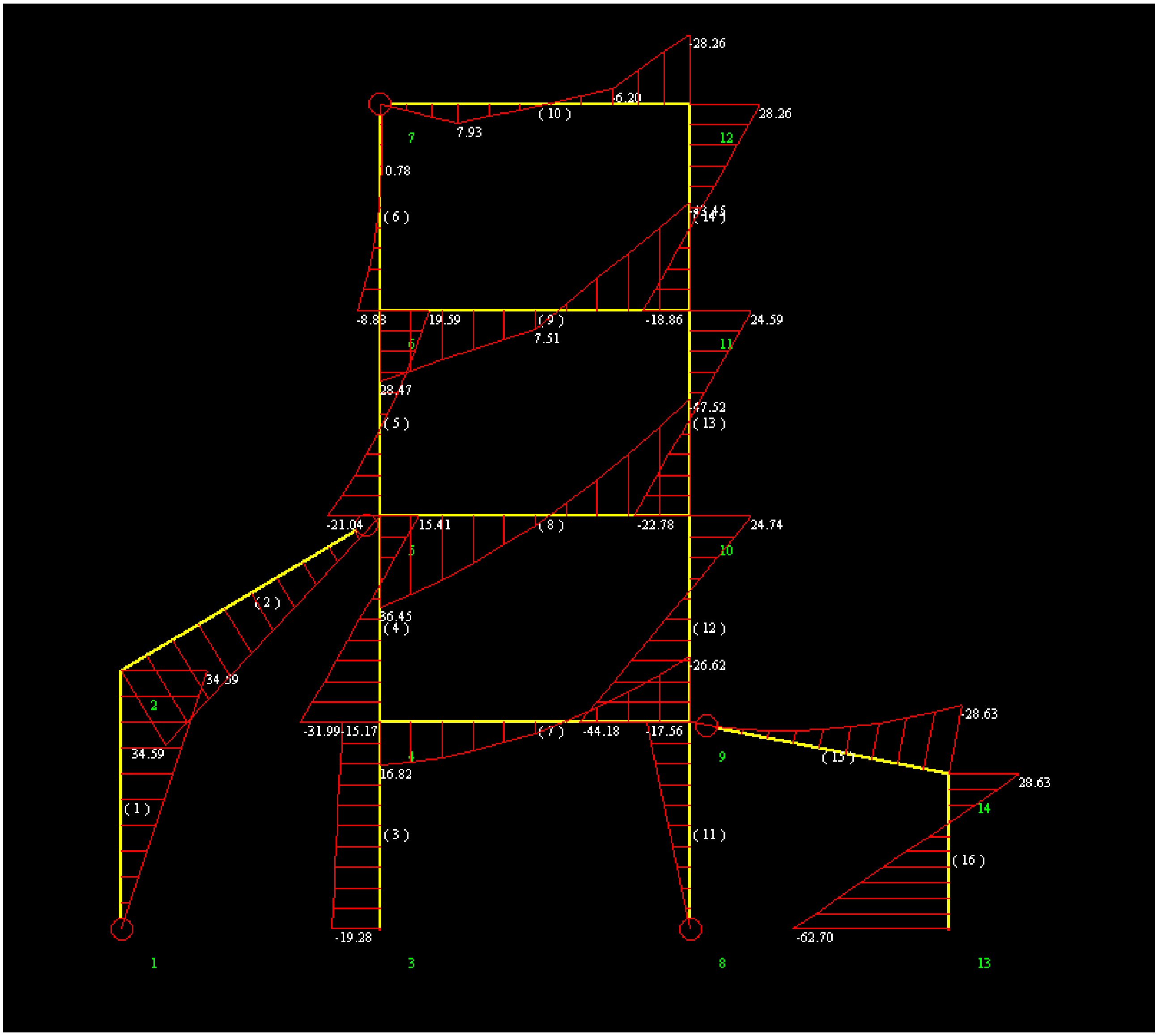

图 4: 弯矩图

A 源程序设计说明

节点类：每个节点有 x, y 坐标以及整体位移编码 gdof

main.py

```
class Joint():
    '''这是节点类'''
    def __init__(self,x,y,GDOF):
        '''
        para::
            x: 节点的x坐标
            y: 节点的y坐标
            GDOF: 节点的整体位移编号，是一个三维向量
```

```
        '''
        self.x = x
        self.y = y
        self.gdof = GDOF
```

单元类：每个节点有杆端节点的编号 jOintNo，整体位移编码 glbdof，杆长 len，与坐标轴正向夹角的余弦值 cos，与坐标轴正向夹角的正弦值 sin，抗弯刚度 ei，抗拉刚度 ea，质量 m

main.py

```
class Element():
    '''这是单元类'''
    def __init__(self,JointNo,GlbDOF,Length,CosA,SinA,EI,EA,mass):
        '''
        para::
            JointNo：两端节点的编号，为二维的向量
            GlbDOF：单元的整体自由度编号，每个节点有三个编号，故共六个，为六维向量
            Length：单元的长度
            CosA：单元与整体x方向夹角的余弦值，在SetElemProp中计算
            SinA：单元与整体x方向夹角的正弦值，在SetElemProp中计算
            EI：单元抗弯刚度
            EA：单元的抗拉刚度
            mass：单元的质量
        '''
        self.jointNo = JointNo
        self.glbdof = GlbDOF
        self.len = Length
        self.cos = CosA
        self.sin = SinA
        self.ei = EI
        self.ea = EA
        self.m = mass
```

节点荷载类：每个节点荷载有节点编号 jointNo，力的作用方向 lodDof，力的大小 lodVal

main.py

```
class JointLoad():
    '''这是节点荷载类'''
    def __init__(self,JointNo,LodDOF,LodVal):
        '''
        para::
            JointNo：节点编号
            LodDOF：荷载的方向，即与哪个方向的自由度对应
            LodVal：荷载的大小
        '''
        self.jointNo = JointNo
        self.lodDof = LodDOF
        self.lodVal = LodVal
```

单元荷载类：每个单元荷载有荷载作用的单元的编号 elemNo，荷载作用的方向 idx，荷载作用的位置 pos，荷载的大小 lodVal

main.py

```
class ElemLoad():
```

```python
    '''这是单元均布荷载类'''
    def __init__(self,ElemNo,Indx,Pos,LodVal):
        '''
        para::
            ElemNo: 单元编号
            Indx: 荷载的方向
            Pos: 荷载的位置
            LodVal: 荷载的大小
        '''
        self.elemNo = ElemNo
        self.idx = Indx
        self.pos = Pos
        self.lodVal = LodVal
```

设置单元属性 `SetElemProp`：输入包含所有单元的数组和包含所有节点的数组，通过将单元的端点与节点向量对应，计算得到杆长和正弦值和余弦值。

main.py

```python
def SetElemProp(Elem, Joint):
    '''
    para::
        Elem: 包含所有单元的数组
        Joint: 包含所有节点的数组
    return:
        Elem
    '''
    Nelem = len(Elem)
    for i in range(Nelem):
        j1 = Elem[i].jointNo[0]
        j2 = Elem[i].jointNo[1]
        Elem[i].glbdof[:3] = Joint[j1-1].gdof
        Elem[i].glbdof[3:] = Joint[j2-1].gdof
        dx = Joint[j2-1].x - Joint[j1-1].x
        dy = Joint[j2-1].y - Joint[j1-1].y
        Elem[i].len = np.sqrt((dx)**2+(dy)**2)
        Elem[i].cos = dx/Elem[i].len
        Elem[i].sin = dy/Elem[i].len
        Elem[i].mass = 0
    return Elem
```

计算坐标转换矩阵 `TransMatrix`：输入单元与整体坐标系间的夹角的正弦值和余弦值，根据不同问题类型对应的转换矩阵的维度差异计算坐标转换矩阵

main.py

```python
def TransMatrix(CosA,SinA):
    '''
    para::
        CosA: 单元与整体坐标系之间的夹角的余弦值
        SinA: 单元与整体坐标系之间的夹角的余弦值
    return:
        ET: 局部坐标系下的单元坐标转换矩阵
    '''
    ET = np.zeros((6,6))
```

```python
    ET[0,:2] = [CosA, SinA]
    ET[1,:2] = [-SinA, CosA]
    if(ET[0,:].shape[0] > 2):
        ET[2,2] = 1
    if (ET[0,:].shape[0] > 3):
        ET[3:6,3:6] = ET[:3,:3]
    return ET
```

计算位移向量和整体刚度矩阵 `varBandSolv`:

main.py

```python
def varBandSolv(Disp, Kcol, GLoad, row1):
    NCol = len(Kcol[0,:])
    Diag = np.array([Kcol[i,i] for i in range(NCol)])

    for j in range(1,NCol):
        for k in range(row1[j],j-1):
            row_1 = max([row1[j],row1[k]])
            s = np.dot(Kcol[row_1-1:k-1,k],Kcol[row_1-1:k-1,j])
            Kcol[k,j] = Kcol[k,j]-s
       # print( Kcol[row1[j]:j-1,j])
        Kcol[row1[j]-1:j-1,j] = Kcol[row1[j]-1:j-1,j]/Diag[row1[j]-1:j-1]
       # print( Kcol[row1[j]:j-1,j])
        s = np.dot(Diag[row1[j]-1:j-1],Kcol[row1[j]-1:j-1,j]**2)
        Diag[j] = Diag[j] - s
    Kcol += Kcol.T - np.diag(Kcol.diagonal())
    # print(Kcol[:10,:10])

    Disp = GLoad
    for j in range(1,NCol):
        Disp[j] -= np.dot(Kcol[row1[j]-1:j-1,j],Disp[row1[j]-1:j-1])

    Disp = np.true_divide(Disp,Diag)
    for j in range(NCol-1,0,-1):
        Disp[row1[j]-1:j-1] = Disp[row1[j]-1:j-1] - Disp[j]*Kcol[row1[j]-1:j-1,j]
    return (Kcol, Disp)
```

计算整体刚度矩阵 `GStifMat`: 输入整体刚度矩阵，包含所有单元的数组

main.py

```python
def GStifMat(Kcol, Elem):
    '''
    para::
        Kcol: 整体刚度矩阵
        Elem: 包含所有单元的数组
    return:
        Kcol
    '''
    NElem = len(Elem)
    for i in range(NElem):
        # 计算局部坐标系下的单元刚度矩阵
        EK = EStifMat(Elem[i].len,Elem[i].ei,Elem[i].ea)
        # 计算单元坐标转换矩阵
        ET = TransMatrix(Elem[i].cos,Elem[i].sin)
```

```
        # 整体坐标系下的单元刚度矩阵
        EK = np.matmul(np.transpose(ET),np.matmul(EK,ET))
        # 单元定位向量
        ELocVec = Elem[i].glbdof
        for j in range(6):
            JGDOF = ELocVec[j]
            if(JGDOF == 0):
                continue
            # 当节点的位移编码不为0时
            for k in range(len(ELocVec)):
                # 将单元刚度矩阵集成到整体刚度矩阵上
                if((ELocVec[k] > 0) and (ELocVec[k] <= JGDOF)):
                    Kcol[ELocVec[k]-1,JGDOF-1] += EK[k,j]
    return Kcol
```

计算整体荷载向量 `GLoadVec`：输入整体荷载向量，包含所有的单元的数组，节点荷载向量，单元和在向量，包含所有节点的向量

main.py

```
def GLoadVec(GLoad,Elem,JLoad,ELoad,Joint):
    '''
    para::
        GLoad: 整体荷载向量
        Elem: 包含所有单元的数组
        JLoad: 节点荷载向量
        ELoad: 单元荷载向量
        Joint: 包含所有节点的向量
    return:
        GLoad: 整体荷载向量
    '''
    NJLoad = len(JLoad)
    for i in range(NJLoad):
        JGDOF = Joint[JLoad[i].jointNo].gdof[JLoad[i].lodDof]
        GLoad[JGDOF] = GLoad[JGDOF]+JLoad[i].lodVal
    NELoad = len(ELoad)
    for i in range(NELoad):
        ET = TransMatrix(Elem[ELoad[i].elemNo].cos,Elem[ELoad[i].elemNo].sin)
        F0 = EFixendF(ELoad[i].idx,ELoad[i].pos,ELoad[i].lodVal,Elem[ELoad[i].elemNo])
        F0 = np.matmul(np.transpose(ET),F0)
        ELocVec = np.array(Elem[ELoad[i].elemNo].glbdof)
        nonZeroIdx = np.where(ELocVec>0)[0]
        GLoad[ELocVec[nonZeroIdx]-1] += F0[nonZeroIdx]

    return GLoad
```

计算位移向量 `solveDisp`：输入所有单元的数组，所有节点的数组，节点荷载向量，单元荷载向量

main.py

```
def solveDisp(Disp,Elem,Joint,JLoad,ELoad):
    '''
    para::
        Disp: 整体位移向量
        Elem: 包含所有单元的数组
        Joint: 包含所有节点的数组
```

```
        JLoad: 节点荷载向量
        ELoad: 单元荷载向量
    return:
        Kcol: 整体刚度矩阵
        Disp: 整体位移向量
    '''
    NGlbDOF = len(Disp)
    GLoad = np.zeros((NGlbDOF))
    Kcol = np.array([[0]*NGlbDOF]*NGlbDOF)
    (rowIdx, KVal, indPtr, row1) = SetMatBand(Kcol, Elem)
    # 得到整体荷载向量
    GLoad = GLoadVec(GLoad, Elem, JLoad, ELoad, Joint)
    # 得到整体刚度矩阵
    Kcol = GStifMat(Kcol, Elem)
    # 得到整体刚度矩阵和位移向量
    (Kcol, Disp) = varBandSolv(Disp, Kcol, GLoad, row1)
    print(Kcol)
    return (Kcol,Disp)
```

计算单元刚度矩阵 `EStifMat`：输入包含所有单元的数组，单元的抗弯刚度，抗拉刚度。

main.py

```
def EStifMat(ELen,EI,EA):
    '''
    para::
        Elem: 包含所有单元的数组
        EI: 单元的抗弯刚度
        EA: 单元的抗拉刚度
    return:
        EK: 单元刚度矩阵
    '''
    EK = np.array([[0]*6]*6)
    EAL = EA/ELen
    EIL1 = EI/ELen
    EIL2 = EI/(ELen**2)
    EIL3 = EI/(ELen**3)
    EK[0,:] = [EAL,0,0,-EAL,0,0]
    EK[1,:] = [0,12*EIL3,6*EIL2,0,-12*EIL3,6*EIL2]
    EK[2,:] = [0,6*EIL2,4*EIL1,0,-6*EIL2,2*EIL1]
    EK[3,:] = [-EAL,0,0,EAL,0,0]
    EK[4,:] = [0,-12*EIL3,-6*EIL2,0,12*EIL3,-6*EIL2]
    EK[5,:] = [0,6*EIL2,2*EIL1,0,-6*EIL2,4*EIL1]

    return EK
```

计算单元杆端力向量 `EFixendF`：输入力的方向，力的位置，力的大小

main.py

```
def EFixendF(Indx,a,q,Elem):
    '''
    para::
        Indx: 力的方向
        a: 力的位置
        q: 力的大小
```

```python
    return:
        F0: 单元力
    '''
    l = Elem.len
    EI = Elem.ei
    EA = Elem.ea
    F0 = np.zeros((6))
    if(Indx == 1):
        F0[1] = -q*(a*l)/2*(2-2*(a*l)**2/(l**2)+(a*l)**3/(l**3))
        F0[2] = -q*(a*l)**2 /12 *(6-8*(a*l)/l+3*(a*l)**2/(l**2))
        F0[4] = -q*(a*l)**3/(l**2)/2*(2-(a*l)/l)
        F0[5] = q*(a*l)**3/l/12*(4-3*(a*l)/l)
    elif(Indx == 2):
        F0[1] = -q*(l-(a*l))**2/(l**2)*(1+2*(a*l)/l)
        F0[2] = -q*(a*l)*(l-(a*l))**2/(l**2)
        F0[4] = -q*(a*l)**2/(l**2)*(1-2*(l-(a*l))/l)
        F0[5] = q*(a*l)**2*(l-(a*l))/(l**2)
    elif(Indx == 3):
        F0[0] = EA*q/l
        F0[3] = -EA*q/l
    elif(Indx == 4):
        F0[1] = 12*EI*q/(l**3)
        F0[2] = 6*EI*q/(l**2)
        F0[4] = -12*EI*q/(l**3)
        F0[5] = 6*EI*q/(l**2)

    for i in range(F0.shape[0]):
        F0[i] = -F0[i]

    return F0
```

计算单元位移向量 `ElemDisp`：输入整体位移向量，单元

main.py

```python
def ElemDisp(Disp,Elem):
    '''
    para::
        Disp: 整体位移向量
        Elem: 一个单元类
    return:
        EDisp: 单元位移向量
    '''
    ET = TransMatrix(Elem.cos,Elem.sin)
    EDisp = np.zeros((6))
    for i in range(len(Elem.glbdof)):
        if(Elem.glbdof[i] > 0):
            EDisp[i] += Disp[Elem.glbdof[i]-1]
    EDisp = np.matmul(np.transpose(ET),EDisp)

    return EDisp
```

计算单元力向量 `ElemForce`：输入单元的编号，整体位移向量，一个单元，单元荷载

main.py

```python
def ElemForce(ie, Disp,Elem,ELoad):
    '''
    para::
        ie: 单元的编号
        Disp: 整体位移向量
        Elem: 一个单元
        ELoad: 单元荷载
    return:
        EForce: 单元荷载向量
    '''
    ET = TransMatrix(Elem.cos,Elem.sin)
    EK = EStifMat(Elem.len,Elem.ei,Elem.ea)
    EDisp = ElemDisp(Disp,Elem)
    # 总的单元荷载向量
    F = [0]*6
    for i in range(len(ELoad)):
        if(ELoad[i].elemNo == ie):
            # 由一个单元荷载引起的单元荷载向量
            F0 = EFixendF(ELoad[i].idx,ELoad[i].pos,ELoad[i].lodVal,Elem)
            F = F + F0

    EForce = np.matmul(EK,np.matmul(ET,EDisp))-F
    EForce = EForce*np.array([-1,1,1,1,1,1])

    return EForce
```

输入数据 InputData:

main.py

```python
def InputData():
    with open('SM90.IPT','r') as f:
        PropType = f.readline().strip('\n')
        if(int(PropType) != 1):
            return
        [NElem,NJoint,NGlbDOF,NJLoad,NELoad] = f.readline().strip('\n').split(',')
        NElem = int(float(NElem))
        NJoint = int(float(NJoint))
        NGlbDOF = int(NGlbDOF)
        NJLoad = int(NJLoad)
        NELoad = int(NELoad)
        jointList = []
        Elem = []
        JLoad = []
        ELoad = []

        for i in range(NJoint):
            temp = f.readline().strip('\n').split(',')
            x = float(temp[0])
            y = float(temp[1])
            gdof = [int(s) for s in temp[2:]]
            joint = Joint(x,y,gdof)
            jointList.append(joint)

        for i in range(NElem):
            [jointN1,jointN2,EA,EI] = f.readline().strip('\n').split(',')
```

```python
            EA = int(float(EA.replace('d','e')))
            # 全部初始化为0
            elem = Element([int(float(jointN1)),int(float(jointN2))],[],0,0,0,int(float(EI)),EA,0)
            Elem.append(elem)
        if(NJLoad > 0):
            for j in range(NJLoad):
                [jointN,lodDof,lodVal] = f.readline().split(',')
                jointLod = JointLoad(int(jointN),int(lodDof),int(float(lodVal)))
                JLoad.append(jointLod)
        if(NELoad > 0):
            for j in range(NELoad):
                [elemNo, Indx, Pos, LodVal] = f.readline().strip('\n').split(',')
                elemLoad = ElemLoad(int(elemNo),int(Indx),float(Pos),int(float(LodVal)))
                ELoad.append(elemLoad)

    return (NElem, NJoint,NGlbDOF,NJLoad,NELoad,jointList,Elem,JLoad,ELoad)
```

输出数据 **OuputResults**:

main.py

```python
def OutputResult(NElem,Disp,Elem,ELoad):
    with open('SMCAI90.out','w') as f:
        f.write('10 0\n')
        EDisp = [0]*6
        for i in range(NElem):
            EDisp = ElemDisp(Disp, Elem[i])
            for j in range(len(ELoad)):
                if((ELoad[j].idx == 3) and (ELoad[j].elemNo == i+1)):
                    if(ELoad[j].pos == 0):
                        EDisp[0] = EDisp[0] + ELoad[j].lodVal
                    else:
                        EDisp[3] = EDisp[3] + ELoad[j].lodVal
                if((ELoad[j].idx==4) and (ELoad[j].elemNo==i+1)):
                    if(ELoad[j].pos == 0):
                        EDisp[1] = EDisp[1] + ELoad[j].lodVal
                    else:
                        EDisp[4] = EDisp[4] + ELoad[j].lodVal
            for k in range(6):
                if(k < 5):
                    f.write(str(EDisp[k]))
                    f.write(' ')
                else:
                    f.write(str(EDisp[k]))
                    f.write('\n')
        for i in range(NElem):
            EForce = ElemForce(i, Disp, Elem[i], ELoad)
            for k in range(len(EForce)):
                if(k < len(EForce)-1):
                    f.write(str(EForce[k]))
                    f.write(' ')
                else:
                    f.write(str(EForce[k]))
                    f.write('\n')
    return
```

主程序:

main.py

```python
def main():
    (NElem, NJoint,NGlbDOF,NJLoad,NELoad,
            jointList,Elem,JLoad,ELoad) = InputData()
    Elem = SetElemProp(Elem, jointList)

    Disp = [0]*NGlbDOF
    (Kcol,Disp) = solveDisp(Disp,Elem,jointList,JLoad,ELoad)
    OutputResult(NElem,Disp,Elem,ELoad)
    return
```