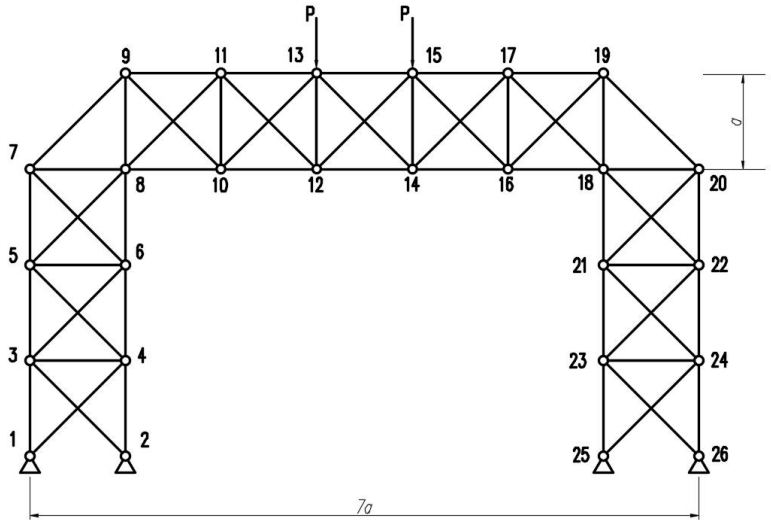
《有限元》大作业
平面桁架结构有限元分析



班级：0118104 班

专业：飞行器设计与工程

成员：**组长**

011810422 包晨宇：myFEM 求解源码编写、统筹分工

组员

011810421 唐天成：计算程序编写、报告撰写

011810424 章乾昱：算例说明、myFEM 求解源码验证

011810425 王奎力：算例编写、myFEM 说明文档编写

011810427 徐飞龙：计算程序编写、流程图及说明图绘制

151830131 吴　限：报告撰写、myFEM 求解源码验证

南京航空航天大学

二〇二一年六月

目录

1 问题重述

1.1 问题描述

如图 1.1 所示，该系统为一平面桁架结构，各个杆在节点处使用铰接，以保证节点处只能传递力而不能传递弯矩。整个系统通过 4 处铰链固定于地面。各个杆的内力也仅有轴力，而杆内没有任何弯矩。

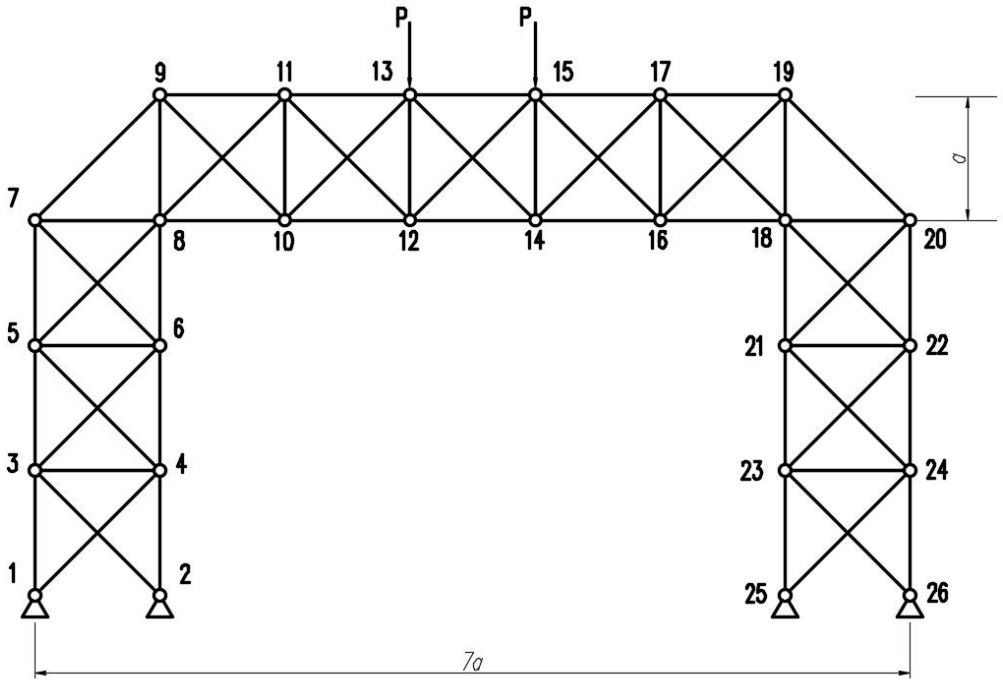


图 1.1 本项目所研究的平面桁架结构

已知在节点 13、15 处有竖直向下的外加载荷 $P = 10kN$，水平杆和竖直杆的长度 $a = 100cm$，各杆截面积 $f = 20cm^2$,材料参数均相同，为弹性模量 $E = 7 \times 10^6 N/cm^2$,泊松比$\mu = 0.3$。需要以此求解各杆轴力，以及各个节点处的位移。

1.2 结构离散化

根据平面桁架受轴力的特点，将它离散为“受轴力的杆单元”。对于本系统的 26 个节点，按照沿着结构传力路径，从左至右顺序标号的原则，对这些节点进行标号。

而对于杆的命名，我们以杆所连接的两个节点将其命名。一共 58 根杆，按照从下至上，从左至右的原则进行标号。例如，连接节点 1 和节点 3 的杆，我们将其命名为 1-3 杆。本系统由 26 个节点和 58 根材料参数，横截面积均相同的杆构成。其结构中各个节点的标号已经列出，各个杆单元由编号“j-i”确认，如杆“1-4”表示由 1 好节点指向 4 号节点的杆。

需要注意的是，这里的连杆编号为有向连杆，遵循“小编号指向大编号”的原则。

1.3 离散杆单元信息表

1.3.1 节点编号

为了展示节点信息，其编号如下表 1.1 节点编号信息表所示。

表 1.1 节点编号信息表

节点	坐标	节点	坐标	节点	坐标	节点	坐标
1	(0,0)	2	(1,0)	3	(0,1)	4	(1,1)
5	(0,2)	6	(1,2)	7	(0,3)	8	(1,3)
9	(1,4)	10	(2,3)	11	(2,4)	12	(3,3)
13	(3,4)	14	(4,3)	15	(4,4)	16	(5,3)
17	(5,4)	18	(5，3)	19	(6，4)	20	(7，3)
21	(6，2)	22	(7，2)	23	(6，1)	24	(7，1)
25	(6，0)	26	(7，0)				

1.3.2 杆编号

本项目对杆采用的编号方法，参考了 Dijsktra 算法和 Floyd 算法中的有向邻接矩阵。在程序中使用 NodeConnection 这一变量，它属于 numpy 数组类型，由 0 和 1 构成。当 j 行 i 列为 1 时，则代表有 j 号节点指向 i 号节点的杆单元连接。这样，我们就可以以一种规范的方式对 58 根杆进行编号，并且方便了后续在 python 中对于连接矩阵 NodeConnection 的建立，并且方便了杆的轴力信息的输出。

1.4 约束声明

对于与支座相连接的节点 1，节点 2，节点 25，节点 26，我们给定了位移约束 u = 0，v = 0，也就是给定了这些节点的位移边界条件。对于载荷作用点的节点 13 和节点 15，我们给定了力的约束$P_x = 0N, P_y = -10kN$,也就是给定了力边界条件。而对于剩余的自由节点，我们给定力的约束$P_x = 0N, P_y = 0$ 而自由节点的水平位移和竖直位移是待定的，需要通过后续方程进行求解。这样，我们就通过给定位移约束和力约束，完成了该平面桁架系统的约束条件的设定。

2 myFEM 有限元求解 python 程序算例

2.1 myFEM 杆系有限元求解框架概述

myFEM 杆系有限元求解框架是由我们组成员独立编写的 python 程序，其可以支持对任意复杂的、二维平面杆系结构问题的处理。其具有如下特色：

1、由组内成员独立编写，调试以及验证，符合书本多个算例，正确性得以保证；

2、内有两个 class 类型，即 Node 节点类与 SysPole 杆系统类，封装好有限元计算步骤；

3、有组内成员编写的较完善的技术文档（见附件）；

4、支持“低代码”编程，即不懂有限元知识的用户也可调用程序完成杆系结构计算；

5、当前可以对任意复杂的、二维平面杆系结构进行处理；

6、支持多种约束条件，包括位移约束、力约束以及弹性支承约束；

7、自动生成计算报告和计算图形。

如图 2.1 中所展示的是 myFEM 程序内部运算情况与用户调用情况的对应图，其较好地展示了 myFEM 杆系有限元求解框架的运作机理。

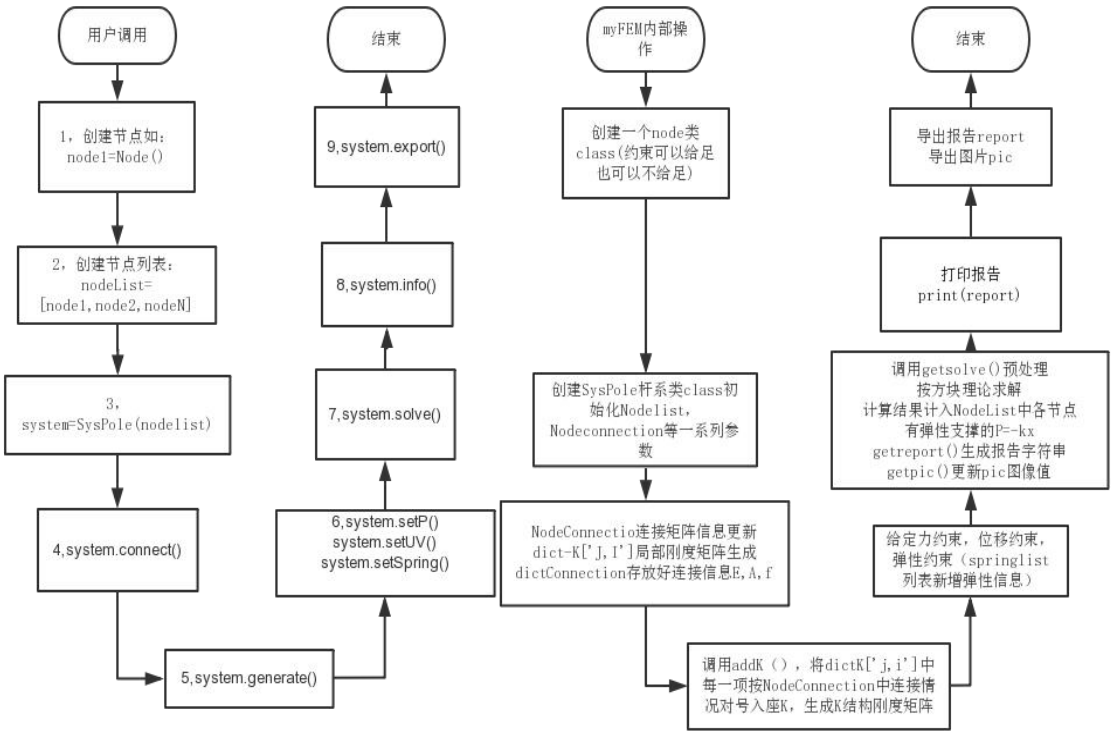


图 2.1 用户操作和 myFEM 内部运算情况流程对应图

在进行有限元求解的过程中，需要由用户先调用 Node 类型创建一些节点，然后导入 SysPole 类中，进行系统组建；组建结束之后，用户可以根据实际需求，调用 SysPole 类中的函数，将各个节点连接起来形成对应的单元刚度矩阵；连接结束后，SysPole 系统会自动根据“对号入座”原则，组装出整体的结构刚度矩阵。

随后给定约束条件，包括力约束和位移约束；值得一提的是，在 myFEM 源码的最新版

本中已支持弹性支承约束；随后系统求解并生成报告、绘制图片。

如图 2.2 为 myFEM 求解框架中，类型的基本变量及函数信息。

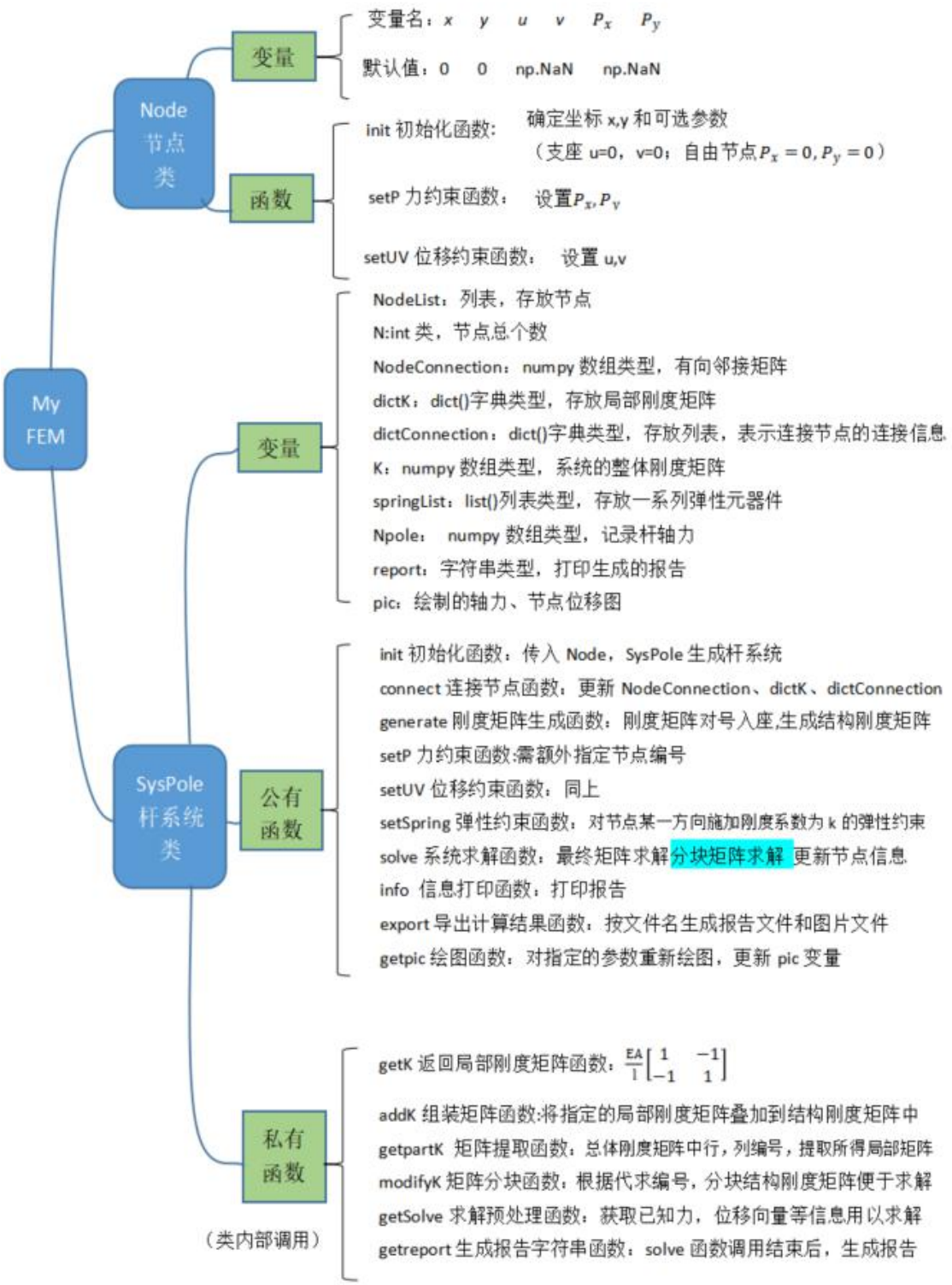


图 2.2 myFEM 求解框架中类型的基本变量及函数信息

2.2 算例描述

为了更好地阐释 myFEM 使用方式，对《飞行器结构力学》一书第二版 P286 页例题 11.6.1 平面桁架结构进行分析，并作算例说明：系统共计四个节点，六根杆，其中 3，4 节点铰支，1 节点受向上的 10kN 的拉力，系统二度静不定。

已知杆弹性模量 $\mathrm{E}=2\times10^{7}\mathrm{N/cm}^{2}$，水平和竖直杆地横截面积为 $1\mathrm{cm}^{2}$，斜的横截面积为$\frac{\sqrt{2}}{2}cm^{2}$，几何尺寸如图 2.3 所示。

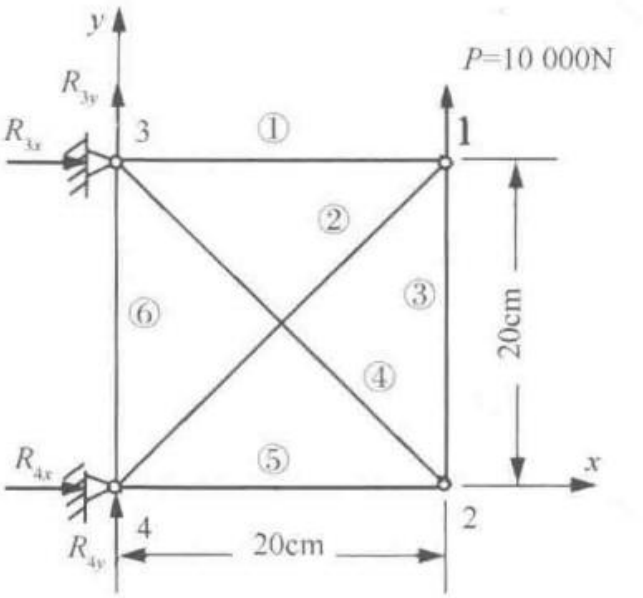


图 2.3 myFEM 平面桁架算例图

2.3 myFEM python 程序求解算例程序

根据 2.1 节所述，用户在调用 myFEM 程序计算时，步骤遵循以下的计算流程图。

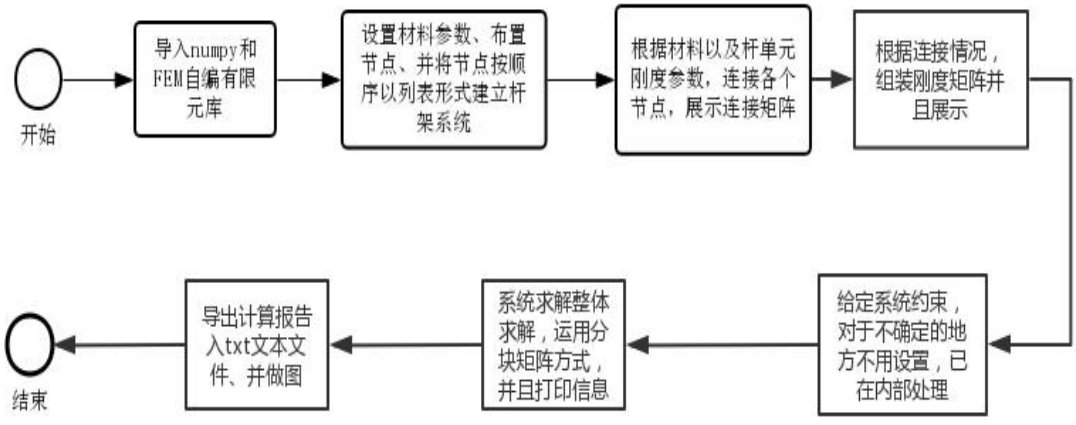


图 2.4 算例求解计算流程图

求解算例的完整程序程序如下：

```
import myFEM as fem

import numpy as np                                  #导入 numpy 和 FEM 自编有限元库

E = 200e9
A1 = 1e-4
l1 = 20e-2
A2 = A1 * np.sqrt(2)/2

l2 = np.sqrt(2) * l1                                #设定材料参数

node1 = fem.Node(l1 , l1)
node2 = fem.Node(l1 , 0)
node3 = fem.Node(0 , l1)

node4 = fem.Node(0 , 0)                             #确定节点位置

nodeList = [node1 , node2 , node3 , node4]

sys = fem.SysPole(nodeList)                         #建立系统

sys.connect(3 , 1 , E , A1)
sys.connect(4 , 1 , E , A2)
sys.connect(1 , 2 , E , A1)
sys.connect(3 , 2 , E , A2)
sys.connect(4 , 2 , E , A1)

sys.connect(3 , 4 , E , A1)                         #连接节点成杆

print('连接矩阵 ： \n' , sys.NodeConnection)

sys.generate()

print(sys.K/0.25/1e6)                               #生成刚度矩阵

sys.setP(1 , Px = 0 , Py = 10e3)
sys.setP(2 , Px = 0 , Py = 0)
sys.setUV(3 , u = 0 , v = 0)

sys.setUV(4 , u = 0 , v = 0)                        #给定约束

sys.solve()
sys.info()

sys.export('demo.txt' , 'demo.png' , 6)             #求解问题及后续报告
```

2.4 算例程序步骤解释

求解算例程序每行含义如下所述。

```
'import myFEM as fem
Import nunpy as np'
```

导入自己编写的有限元方法计算库‘myFEM’以及 python 的通用计算库 numpy。

```
'E = 200e9
A1 = 1e-4
l1 = 20e-2
A2 = A1 * np.sqrt(2)/2
l2 = np.sqrt(2) * l1'
```

设定基本参数：杆件弹性模量 $\mathrm{E}=200\mathrm{e}9\mathrm{N}/\mathrm{m}^2$，水平杆和垂直杆横截面积$\mathrm{A}_1=1\mathrm{e}--4\mathrm{m}^2$，水平杆和垂直杆杆长$l_1=0.2m$，斜杆横截面积为横平竖直杆的$\frac{\sqrt{2}}{2}$倍，斜杆杆长为横平竖直杆长的$\sqrt{2}$倍。

```
'node1 = fem.Node(l1 , l1)'
'node2 = fem.Node(l1 , 0)'
'node3 = fem.Node(0 , l1)'
'node4 = fem.Node(0 , 0)'
'nodeList = [node1 , node2 , node3 , node4]'
```

建立四个节点类型，并做成一个 list 列表。

```
'sys = fem.SysPole(nodeList)'
```

运用列表‘nodeList’中的信息，建立 SysPole 系统‘sys’。随即调用初始化函数，传入一个全是 Node 的 list 列表，从而对系统进行初始化，生成如下变量（具体细节见附件中的源程序，以及说明文档）：

1. NodeList：存放着节点顺序表；
2. N：为节点个数，共 4 个节点；
3. dictK：字典类型，用以存放各种名称的部分刚度矩阵，暂时为空；
4. dictConnection：字典类型，用以存放某个连接的连接信息，暂时为空
5. K：总体刚度矩阵，暂时全为 0；
6. springList：列表类型，弹性元器件信息表，暂时为空；
7. report：生成报告字符串，暂时只有标题。

```
'sys.connect(3 , 1 , E , A1)'
'sys.connect(4 , 1 , E , A2)'
'sys.connect(1 , 2 , E , A1)'
'sys.connect(3 , 2 , E , A2)'
'sys.connect(4 , 2 , E , A1)'
'sys.connect(3 , 4 , E , A1)'
```

实现节点的连接，共计六根杆；这里需要说明的是，算法中以矩阵形式表示节点连接形成杆件情况——本部分利用了 connect 函数对系统中的节点进行连接，连接系统中节点，需要给定以下参数：

j：连接起点

i：连接终点

E：连接杆的弹性模量

A：横截面积

在 myFEM 求解框架的内部中，会进行如下操作：

1. 将 NodeConnection 的 j 行 i 列置为 1，表明杆由 j 向 i 连接；
2. 对 dictConnection 连接信息字典中的['j,i']键值内添加列表[E,A,l]，其中 l 为杆长，直接根据节点相对位置计算：$l=\sqrt{(node1.x-node2.x)^2+(node1.y-node2.y)^2}$；
3. 对 dictK 局部刚度矩阵，[‘j，i’]键值设为计算而得的局部刚度矩阵，调用了__getK( )私有函数。

连接完成后，NodeConnection 更新信息表示如下：

$$\mathbf{sys.NodeConnection}=\begin{bmatrix}0&1&0&0\\0&0&0&0\\1&1&0&1\\1&1&0&0\end{bmatrix}$$

4×4 矩阵中第 j 行 i 列数据为 1 即表示为第 j 节点和第 i 节点之间有连接(此处只考虑了规定点到点方向的连接故此不是一个对称矩阵)。

而 dictK 部分单元局部刚度矩阵更新如下。

节点 1，2 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['1,2']}=\begin{bmatrix}0&0&0&0\\0&1&0&-1\\0&0&0&0\\0&-1&0&1\end{bmatrix}\times 10^8$$

而节点 2，3 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['2,3']}=\begin{bmatrix}0.25&-0.25&-0.25&0.25\\-0.25&0.25&0.25&-0.25\\-0.25&0.25&0.25&-0.25\\0.25&-0.25&-0.25&0.25\end{bmatrix}\times 10^8$$

‘sys.generate()’

生成系统结构刚度矩阵，按照‘对号入座’方法组装刚度矩阵。

K 刚度矩阵信息更新如下：

$$\mathbf{sys.K}=\begin{bmatrix}500&100&0&0&-400&0&-100&-100\\100&500&0&-400&0&0&-100&-100\\0&0&500&-100&-100&100&-400&0\\0&-400&-100&500&100&-100&0&0\\-400&0&-100&100&500&-100&0&0\\0&0&100&-100&-100&500&0&-400\\-100&-100&-400&0&0&0&500&100\\-100&-100&0&0&0&400&100&500\end{bmatrix}\times 0.25\times 10^6$$

‘sys.setP(1 , Px = 0 , Py = 10e3)’

‘sys.setP(2 , Px = 0 , Py = 0)’

‘sys.setUV(3 , u = 0 , v = 0)’

‘sys.setUV(4 , u = 0 , v = 0) ‘

给定各个节点的约束，比如给 1 节点横向无载荷，纵向 10kN 载荷；2 节点自由节点，无外载荷；3 节点和 4 节点都和支座相连，位移为零。

‘sys.solve()’

调用函数求解未知量，这里使用了分块矩阵求解的方法。具体实现情况参考附件中的说明文档以及 myFEM 程序源码。

‘sys.info()’

打印所需各个信息报告，如连接矩阵，刚度矩阵，各点受力及位移，各杆轴力。

'sys.export('demo.txt' , 'demo.png' , 6) '

导出计算报告为指定的 txt 文本，导出绘制的图片文件。

2.5 报告信息以及变量展示

2.5.1 节点连接矩阵

当前 SysPole 类系统下的连接矩阵 NodeConnection 数据如下，表明了各个节点间的连接情况。

$$\mathbf{sys.NodeConnection} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

2.5.2 刚度矩阵

结构刚度矩阵，即拼装后的系统总刚度矩阵如下：

$$\mathbf{sys.K} = \begin{bmatrix} \mathbf{500} & \mathbf{100} & \mathbf{0} & \mathbf{0} & \mathbf{-400} & \mathbf{0} & \mathbf{-100} & \mathbf{-100} \\ \mathbf{100} & \mathbf{500} & \mathbf{0} & \mathbf{-400} & \mathbf{0} & \mathbf{0} & \mathbf{-100} & \mathbf{-100} \\ \mathbf{0} & \mathbf{0} & \mathbf{500} & \mathbf{-100} & \mathbf{-100} & \mathbf{100} & \mathbf{-400} & \mathbf{0} \\ \mathbf{0} & \mathbf{-400} & \mathbf{-100} & \mathbf{500} & \mathbf{100} & \mathbf{-100} & \mathbf{0} & \mathbf{0} \\ \mathbf{-400} & \mathbf{0} & \mathbf{-100} & \mathbf{100} & \mathbf{500} & \mathbf{-100} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{100} & \mathbf{-100} & \mathbf{-100} & \mathbf{500} & \mathbf{0} & \mathbf{-400} \\ \mathbf{-100} & \mathbf{-100} & \mathbf{-400} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{500} & \mathbf{100} \\ \mathbf{-100} & \mathbf{-100} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{400} & \mathbf{100} & \mathbf{500} \end{bmatrix} \times \mathbf{0.25} \times \mathbf{10^6}$$

局部的单元刚度矩阵部分展示如下所述。

节点 1，2 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['1,2']} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{bmatrix} \times \mathbf{10^8}$$

而节点 2，3 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['2,3']} = \begin{bmatrix} \mathbf{0.25} & \mathbf{-0.25} & \mathbf{-0.25} & \mathbf{0.25} \\ \mathbf{-0.25} & \mathbf{0.25} & \mathbf{0.25} & \mathbf{-0.25} \\ \mathbf{-0.25} & \mathbf{0.25} & \mathbf{0.25} & \mathbf{-0.25} \\ \mathbf{0.25} & \mathbf{-0.25} & \mathbf{-0.25} & \mathbf{0.25} \end{bmatrix} \times \mathbf{10^8}$$

节点 3，4 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['3,4']} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{-1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{bmatrix} \times \mathbf{10^8}$$

节点 2，4 组成杆件局部刚度矩阵有：

$$\mathbf{sys.dictK['2,4']} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-1} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \times \mathbf{10^8}$$

2.5.3 轴力矩阵

轴力矩阵 Npole 参考了连接矩阵 NodeConnection 的定义，第 j 行第 i 列数值即代表 j，i 两节点所形成杆件的轴力信息，绝对值为大小，正为拉力负为压力，其信息如下：

$$\mathbf{sys.Npole} = \begin{bmatrix} \mathbf{0} & \mathbf{4545.45454545} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{-5454.54545455} & \mathbf{-6428.24346533} & \mathbf{0} & \mathbf{0} \\ \mathbf{7713.8921584} & \mathbf{-6428.24346533} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

2.5.4 各个节点信息

将各个节点的信息做成表格，如下 2.1 节点信息表所示。

表 2.1 节点信息表

节点编号	位置坐标 x/m	位置坐标 y/m	受力 Px/N	受力 Py/N	位移 u/m	位移 v/m
1	0.2	0.2	0	10000	-0.00005454545	0.00027272727
2	0.2	0	0	0	0.000045454545	0.00027272727
3	0	0.2	10000	-4545.4545	0	0
4	0	0	-10000	-5454.5454	0	0

2.5.5 各杆轴力计算结果图

由轴力矩阵 Npole 所显示的信息，可以表示各个杆的轴力，如表 2.2 所示。

表 2.2 轴力结果表

杆件	1-2	1-3	1-4	2-3	2-4	3-4
轴力	4545.454545N	-5454.5454N	7713.8921N	-6428.2434N	4545.4545N	0

在 myFEM 程序中的 solve()函数调用下，会经由 matplotlib 绘图可视化函数库，根据节点信息、计算求解的位移结果、轴力等自动绘制一张系统二维图片，并保存在 SysPole 类的 pic 变量类型中。

该算例在’sys.solve()’命令后，也产生了如下的图片信息。

输出结果如图所示，其中灰色字体为节点编号，黑色字体为轴力信息，蓝色字体为节点位移状态。

不难发现，该结果的位移边界条件符合，并且在 1 节点和 2 节点上的受力是平衡的；同时也发现《飞行器结构力学》一书对于该算例的轴力图表示有误。

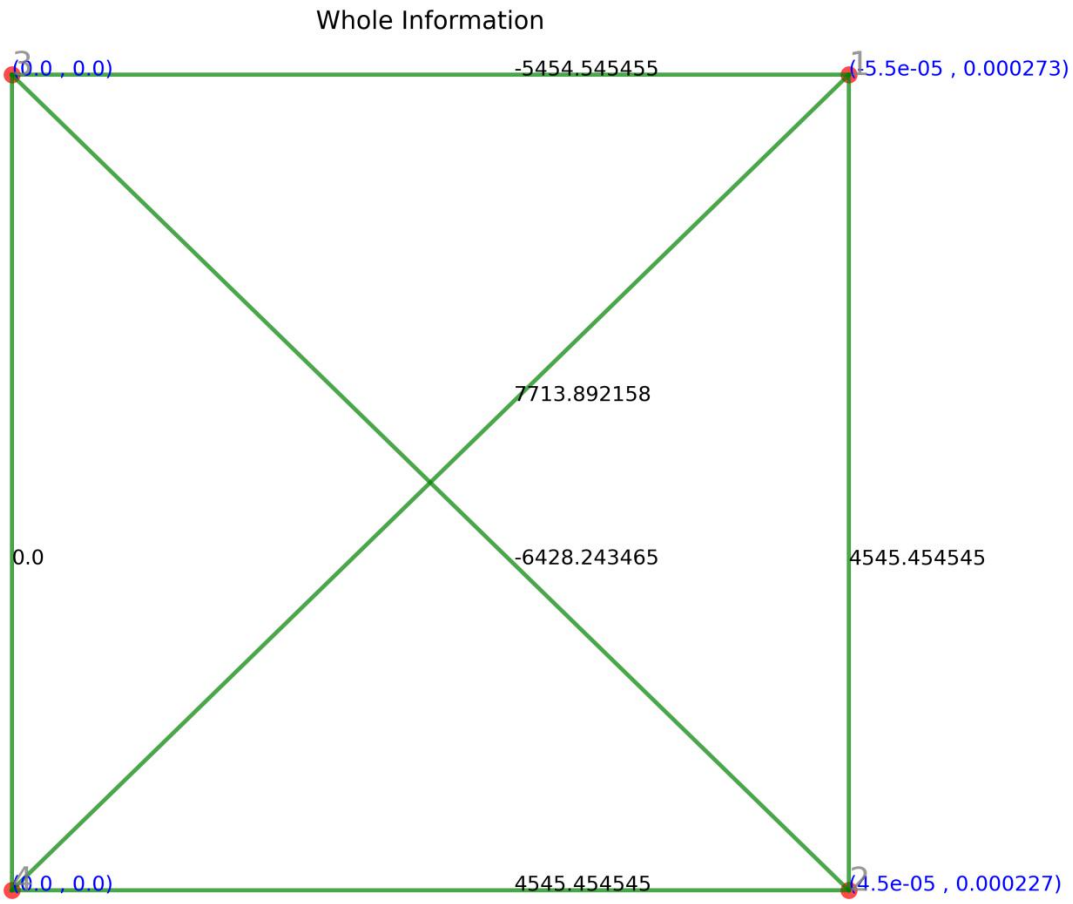


图 2.5 轴力和位移结果输出

3 基于 myFEM 求解框架的问题解决

3.1 节点信息录入及系统建立

```
import numpy as np
import matplotlib.pyplot as plt
import myFEM as fem
```

依次导入 numpy 计算库，matplotlib 绘图库，再导入我们自己编写的有限元方法计算库。为了不让变量冲突，不全部导入。

```
P = 10e3
a = 100e-2
f = 20e-4
E = 7e6 * 10**4
mu = 0.3
```

录入 13、15 节点处外载荷 $P = 10000N$，边长 $a = 100cm$，各杆截面积 $f = 20cm^2$，材料弹性模量 $E = 7 \times 10^6 N/cm^2$，泊松比 $\mu = 0.3$。

```
nodeList = list()
```

增设一个空列表 nodelist，以便后续将各个节点添加到此列表中。

```
for i in range(4):
    tmp1 = fem.Node(0 , i * a , Px = 0 , Py = 0)
    tmp2 = fem.Node(a , i * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
pass
```

将 1-8 号节点信息依次录入节点列表中。由于奇数节点横坐标均为 0，纵坐标从 0 开始依次增加 a，偶数节点横坐标均为 a，纵坐标从 0 开始依次增加 a，因此由以上循环可将 1-8 号节点坐标依次录入。

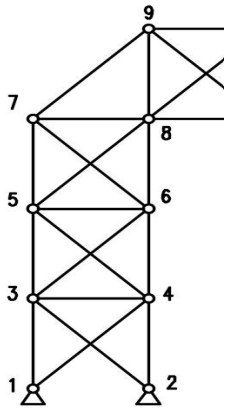


图 3.1 1–8 号节点示意图

```
nodeList.append( fem.Node(a , 4*a , Px = 0 , Py = 0) )
```

将 9 号节点信息录入节点列表中。

```
for i in range(5):
    tmp1 = fem.Node( (2 + i) * a , 3 * a , Px = 0 , Py = 0)
    tmp2 = fem.Node( (2 + i) * a , 4 * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
pass
```

将 10-19 号节点信息依次录入节点列表中。由于偶数节点纵坐标均为 3a，横坐标从 2a 开始依次增加 a，奇数节点纵坐标均为 4a，横坐标从 2a 开始依次增加 a，因此由以上循环可将 10-19 号节点坐标依次录入。

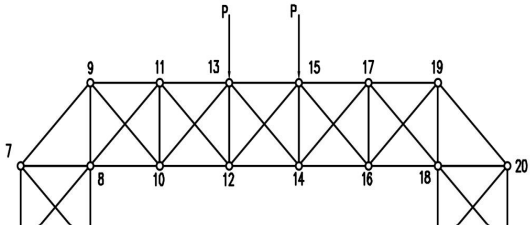


图 3.2 10–19 号节点示意图

```
nodeList.append( fem.Node(7 * a , 3 * a , Px = 0 , Py = 0) )
```

将 20 号节点信息录入节点列表中。

```
for i in range(3):
    tmp1 = fem.Node( 6 * a , (2 - i) * a , Px = 0 , Py = 0)
    tmp2 = fem.Node( 7 * a , (2 - i) * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
    pass
```

将 21-26 号节点信息录入节点列表中。由于奇数节点横坐标均为 6a，纵坐标从 2a 开始依次减少 a，偶数节点横坐标均为 7a，纵坐标从 2a 开始依次减少 a，因此由以上循环可将 21-26 号节点坐标依次录入。

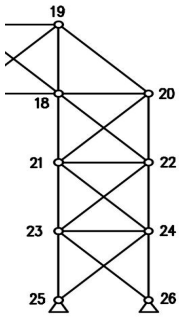


图 3.3 21-26 号节点示意图

```
sys = fem.SysPole(nodeList)
```

初始化函数，传入一个全是 Node 的 list 列表，从而对系统进行初始化，生成如下变量：

1. NodeList：存放着节点顺序表；
2. N：为节点个数，共 26 个节点；
3. NodeConnection：为连接矩阵，有向，暂时全为 0；
4. dictK：字典类型，用以存放各种名称的部分刚度矩阵，暂时为空；
5. dictConnection：字典类型，用以存放某个连接的连接信息，暂时为空；
6. K：组装刚度矩阵，暂时全为 0；
7. springList：弹性元器件列表，存储着弹性元器件存放列表，暂时为空；
8. report：生成报告字符串，暂时只有标题。

3.2 约束条件

```
for i in [1 , 2 , 25 , 26]:
    sys.setUV(i , u = 0 , v = 0)
    sys.setP(i , Px = np.NaN , Py = np.NaN)
Pass
```

由于节点 1、2、25、26 处为支座，因此将四个节点给定约束 $u = 0$ 且 $v = 0$，P_x与P_y为待求量。

```
for i in [13 , 15]:
    sys.setP(i , Px = 0 , Py = - P)
Pass
```

由于节点 13、15 处有 $P = 10000N$，且方向竖直向下，因此将两个节点给定约束$P_x = 0$，$P_y = -10000$。

```
del i , tmp1 , tmp2
```

清除函数中的无用变量。

3.3 节点连接及结构刚度矩阵展示

本部分利用了 connect 函数对系统中的节点进行连接。

连接系统中节点，需要给定以下参数：

j：连接起点

i：连接终点

E：连接杆的弹性模量

A：横截面积

在 myFEM 框架内部，会进行如下操作：

1. 将 NodeConnection 的 j 行 i 列置为 1，表明杆由 j 向 i 连接；
2. 对 dictConnection 连接信息字典中的[‘j，i’]键值内添加列表[E,A,l]，其中 l 为杆长，直接根据节点相对位置计算而成；
3. 对 dictK 局部刚度矩阵，[‘j，i’]键值设为计算而得的局部刚度矩阵，调用了__getK( )私有函数。

```
for i in [1 , 3 , 5 , 8 , 10 , 12 , 14 , 16]:
    sys.connect(i , i+2 , E , f)
    sys.connect(i , i+3 , E , f)
    sys.connect(i+1 , i+2 , E , f)
    sys.connect(i+1 , i+3 , E , f)
    sys.connect(i+2 , i+3 , E , f)
pass
```

对于节点 1、2、3、4 所组成的小系统分析，1-3，1-4，2-3，2-4，3-4 由杆连接，因此若四根杆编号分别为 i、i+1、i+2、i+3，则 i 与 i+2，i 与 i+3，i+1 与 i+2，i+1 与 i+3，i+2 与 i+3 由杆连接。因此当 i 取 1 , 3 , 5 , 8 , 10 , 12 , 14 , 16 时，满足以上所述要求，可将 40 根杆相连。每根杆弹性模量均为 E，截面面积均为 f。

```
for i in [26 , 24]:
    sys.connect(i-2 , i , E , f)
    sys.connect(i-3 , i , E , f)
    sys.connect(i-2 , i-1 , E , f)
    sys.connect(i-3 , i-1 , E , f)
    sys.connect(i-3 , i-2 , E , f)
    pass
```

对于节点 26、25、24、23 所组成的小系统分析，24-26，23-26，24-25，23-25，23-24 由杆连接，因此若四根杆编号分别为 i、i-1、i-2、i-3，则 i-2 与 i，i-3 与 i，i-2 与 i-1，i-3 与 i-1，i-3 与 i-2 由杆连接。因此当 i 取 24、26 时，满足以上所述要求，可将 10 根杆相连。每根杆弹性模量均为 E，截面面积均为 f。

```
sys.connect(7 , 9 , E , f)
sys.connect(8 , 9 , E , f)
sys.connect(19 , 20 , E , f)
sys.connect(18 , 20 , E , f)
sys.connect(18 , 22 , E , f)
sys.connect(18 , 21 , E , f)
sys.connect(20 , 22 , E , f)
sys.connect(20 , 21 , E , f)
```

对于以上 50 杆之外的其他 8 根杆，单独连接。每根杆弹性模量均为 E，截面面积均为 f。

至此系统中 58 根杆全部录入，所有杆的节点编号均为小编号节点在前，大节点编号在后。

```
del i
```

清除函数中的无用变量。

sys.generate()

在连接结束系统之后，可以获得系统的整体刚度矩阵，把 dictK 中所有的局部刚度矩阵叠加到总体刚度矩阵 K 中，就获得了最终的刚度矩阵。

刚度矩阵展示：

由于整个系统为轴对称结构，因此下面选取几组对称杆件进行刚度矩阵展示。

1.“1-3 杆”与“24-26 杆”刚度矩阵展示：

```
In [13]: sys.dictK['1,3']

Out[13]: array([[ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
                [ 0.0e+00,  1.4e+08,  0.0e+00, -1.4e+08],
                [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
                [ 0.0e+00, -1.4e+08,  0.0e+00,  1.4e+08]])
```

图 3.4 代码示意图 1

$$\text{sys.dictK['1,3']} = \mathrm{K}_{13} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1.4 & 0 & -1.4 \\ 0 & 0 & 0 & 0 \\ 0 & -1.4 & 0 & 1.4 \end{bmatrix} \times 10^8$$

```
In [14]: sys.dictK['24,26']

Out[14]: array([[ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
                [ 0.0e+00,  1.4e+08,  0.0e+00, -1.4e+08],
                [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
                [ 0.0e+00, -1.4e+08,  0.0e+00,  1.4e+08]])
```

图 3.5 代码示意图 2

$$\text{sys.dictK['24,26']} = \mathrm{K}_{2426} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1.4 & 0 & -1.4 \\ 0 & 0 & 0 & 0 \\ 0 & -1.4 & 0 & 1.4 \end{bmatrix} \times 10^8$$

2.“1-4 杆”与“23-26 杆”刚度矩阵展示：

```
In [5]: sys.dictK['1,4']

Out[5]: array([[ 49497474.68305831,  49497474.68305831, -49497474.68305831,
                -49497474.68305831],
               [ 49497474.68305831,  49497474.68305831, -49497474.68305831,
                -49497474.68305831],
               [-49497474.68305831, -49497474.68305831,  49497474.68305831,
                 49497474.68305831],
               [-49497474.68305831, -49497474.68305831,  49497474.68305831,
                 49497474.68305831]])
```

图 3.6 代码示意图 3

$$\text{sys.dictK['1,4']} = \mathrm{K}_{14} = \begin{bmatrix} 0.495 & 0.495 & -0.495 & -0.495 \\ 0.495 & 0.495 & -0.495 & -0.495 \\ -0.495 & -0.495 & 0.495 & 0.495 \\ -0.495 & -0.495 & 0.495 & 0.495 \end{bmatrix} \times 10^8$$

```
In [7]: sys.dictK['23,26']

Out[7]: array([[ 49497474.68305831, -49497474.68305831, -49497474.68305831,
                 49497474.68305831],
               [-49497474.68305831,  49497474.68305831,  49497474.68305831,
                -49497474.68305831],
               [-49497474.68305831,  49497474.68305831,  49497474.68305831,
                -49497474.68305831],
               [ 49497474.68305831, -49497474.68305831, -49497474.68305831,
                 49497474.68305831]])
```

图 3.7 代码示意图 4

$$\text{sys.dictK['23,26']} = \mathrm{K}_{2326} = \begin{bmatrix} 0.495 & -0.495 & -0.495 & 0.495 \\ -0.495 & 0.495 & 0.495 & -0.495 \\ -0.495 & 0.495 & 0.495 & -0.495 \\ 0.495 & -0.495 & -0.495 & 0.495 \end{bmatrix} \times 10^8$$

3.“9-11 杆”与“17-19 杆”刚度矩阵展示：

```
In [5]: sys.dictK['9,11']

Out[5]: array([[ 1.4e+08,  0.0e+00, -1.4e+08,  0.0e+00],
               [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
               [-1.4e+08,  0.0e+00,  1.4e+08,  0.0e+00],
               [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00]])
```

图 3.8 代码示意图 5

$$\text{sys.dictK['9,11']} = \mathrm{K}_{911} = \begin{bmatrix} 1.4 & 0 & -1.4 & 0 \\ 0 & 0 & 0 & 0 \\ -1.4 & 0 & 1.4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times 10^8$$

```
In [7]: sys.dictK['17,19']

Out[7]: array([[ 1.4e+08,  0.0e+00, -1.4e+08,  0.0e+00],
               [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00],
               [-1.4e+08,  0.0e+00,  1.4e+08,  0.0e+00],
               [ 0.0e+00,  0.0e+00,  0.0e+00,  0.0e+00]])
```

图 3.9 代码示意图 6

$$\text{sys.dictK['17,19']} = \mathrm{K}_{1719} = \begin{bmatrix} 1.4 & 0 & -1.4 & 0 \\ 0 & 0 & 0 & 0 \\ -1.4 & 0 & 1.4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times 10^8$$

4.“9-10 杆”与“16-19 杆”刚度矩阵展示：

```
In [8]: sys.dictK['9,10']

Out[8]: array([[ 49497474.68305831, -49497474.68305831, -49497474.68305831,
                 49497474.68305831],
               [-49497474.68305831,  49497474.68305831,  49497474.68305831,
                -49497474.68305831],
               [-49497474.68305831,  49497474.68305831,  49497474.68305831,
                -49497474.68305831],
               [ 49497474.68305831, -49497474.68305831, -49497474.68305831,
                 49497474.68305831]])
```

图 3.10 代码示意图 7

$$\text{sys.dictK['9,10']} = \mathrm{K}_{910} = \begin{bmatrix} 0.495 & -0.495 & -0.495 & 0.495 \\ -0.495 & 0.495 & 0.495 & -0.495 \\ -0.495 & 0.495 & 0.495 & -0.495 \\ 0.495 & -0.495 & -0.495 & 0.495 \end{bmatrix} \times 10^8$$

```
In [9]: sys.dictK['16,19']

Out[9]: array([[ 49497474.68305831,  49497474.68305831, -49497474.68305831,
                -49497474.68305831],
               [ 49497474.68305831,  49497474.68305831, -49497474.68305831,
                -49497474.68305831],
               [-49497474.68305831, -49497474.68305831,  49497474.68305831,
                 49497474.68305831],
               [-49497474.68305831, -49497474.68305831,  49497474.68305831,
                 49497474.68305831]])
```

图 3.11 代码示意图 8

$$\text{sys.dictK['16,19']} = \mathrm{K}_{1619} = \begin{bmatrix} 0.495 & 0.495 & -0.495 & -0.495 \\ 0.495 & 0.495 & -0.495 & -0.495 \\ -0.495 & -0.495 & 0.495 & 0.495 \\ -0.495 & -0.495 & 0.495 & 0.495 \end{bmatrix} \times 10^8$$

3.4 系统求解及报告展示

```
sys.solve()
sys.getpic(ftsize = 15)
sys.export('hw.txt' , 'hw.png')
```

将系统求解，导出报告‘hw.txt’以及图片’hw.png’。

1. 验证整体受力是否平衡

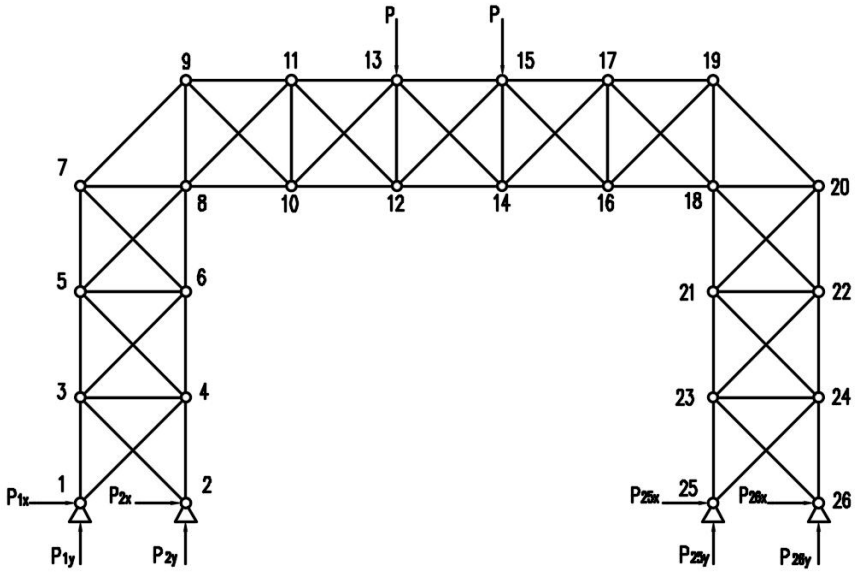


图 3.12 整体受力验算图

由节点信息验算：支座 1，2，25，26 支反力和 13，15 节点处外载荷，应有：

$$P_{1x} + P_{2x} + P_{25x} + P_{26x} = 0$$

$$P_{1y} + P_{2y} + P_{25y} + P_{26y} - 2P = 0$$

代入数据，得：

$$3064.09 + 931.61 - 931.61 - 3064.09 = 0$$

$$8722.41 + 1277.59 + 1277.59 + 8722.41 - 2 \times 10000 = 0$$

即满足该系统受力平衡。

2. 节点连接矩阵

用 NodeConnection 函数形成一个节点连接矩阵，由 0 和 1 构成，其 j 行 i 列为 1，则表示有 j 号节点指向 i 号节点的杆单元连接。

例如对于 1-6 号节点，相应的节点连接矩阵如下：

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

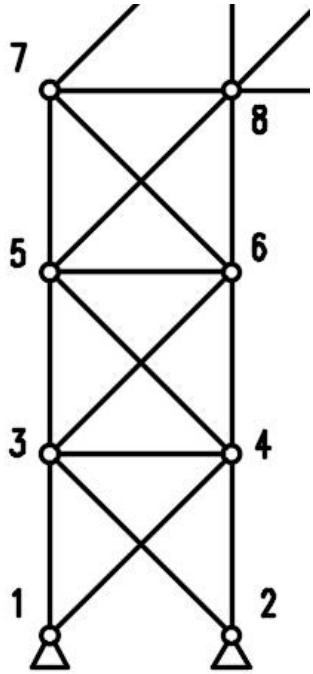


图 3.13 节点连接矩阵局部展示示意图

则表示 1 号节点与 3 号、4 号节点相连，2 号节点与 3 号、4 号节点相连，3 号节点与 4 号、5 号、6 号节点相连，4 号节点与 5 号、6 号节点相连，5 号节点与 6 号节点相连。

本矩阵对于一根杆只有一个对应元素为 1，例如 1-3 杆，只有第 1 行第 3 列元素为 1，而第 3 行第 1 列元素为 0，不重复。

3. 轴力矩阵

用 Npole 函数形成一个轴力矩阵，j 行 i 列上记录着 j 连接 i 的杆上轴力。

例如求 1-3 杆轴力，则输入程序“*sys.Npole[1-1][3-1]*”，显示为

```
In  [11]: sys.Npole[1-1][3-1]

Out[11]: -5658.319723074775
```

图 3.14 代码展示图 9

表示 1-3 杆上轴力为-5658.32N。又如求 19-20 杆轴力，则输入程“*sys.Npole[19-1][20-1]*”，显示为：

```
In  [13]: sys.Npole[19-1][20-1]

Out[13]: 4616.983342765438
```

图 3.15 代码展示图 10

则表示 19-20 杆上轴力为 4616.98N。

由程序自动绘制的轴力图及节点位移图如下图所示，在附件中也有原件，像素较高。

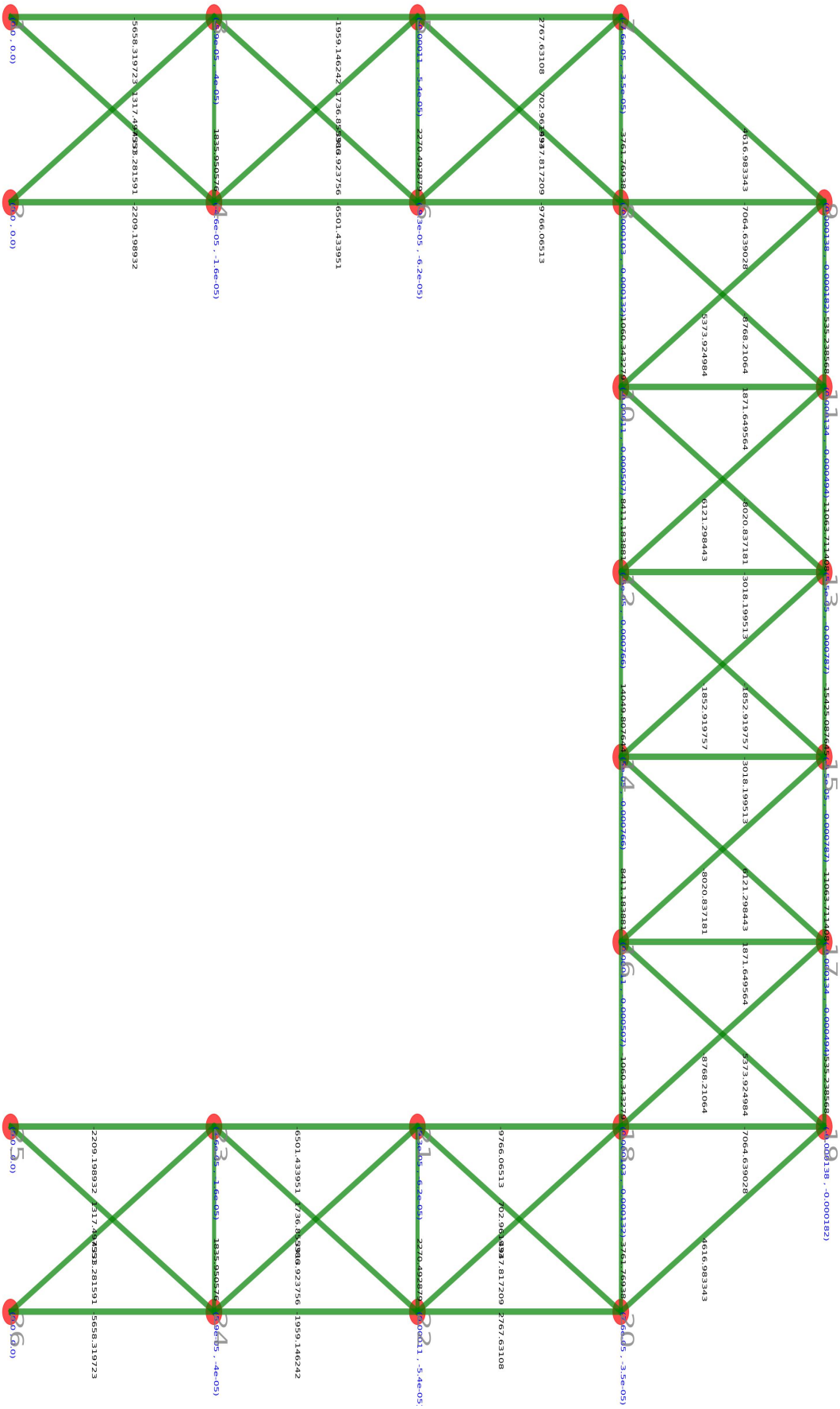


图 3.15 轴力图及节点位移图

3.5 各点信息及轴力展示

各节点信息如下表所示：

表 3.1 各节点信息

编号	x 坐标	y 坐标	水平位移	垂直位移	水平受载	垂直受载
1	0	0	0	0	3064.092798	8722.412521
2	1	0	0	0	931.6114527	1277.587479
3	0	1	-5.9238E-05	-4.04166E-05	0	0
4	1	1	-4.6124E-05	-1.578E-05	0	0
5	0	2	-0.000109567	-5.44105E-05	0	0
6	1	2	-9.33489E-05	-6.22188E-05	0	0
7	0	3	-7.58141E-05	-3.46417E-05	0	0
8	1	3	-0.000102684	-0.000131976	0	0
9	1	4	0.000137939	-0.000182438	0	0
10	2	3	-0.000110258	-0.000507405	0	0
11	2	4	0.000134116	-0.000494037	0	0
12	3	3	-5.01779E-05	-0.000765778	0	0
13	3	4	5.50896E-05	-0.000787336	0	-10000
14	4	3	5.01779E-05	-0.000765778	0	0
15	4	4	-5.50896E-05	-0.000787336	0	-10000
16	5	3	0.000110258	-0.000507405	0	0
17	5	4	-0.000134116	-0.000494037	0	0
18	6	3	0.000102684	-0.000131976	0	0
19	6	4	-0.000137939	-0.000182438	0	0
20	7	3	7.58141E-05	-3.46417E-05	0	0
21	6	2	9.33489E-05	-6.22188E-05	0	0
22	7	2	0.000109567	-5.44105E-05	0	0
23	6	1	4.6124E-05	-1.578E-05	0	0
24	7	1	5.9238E-05	-4.04166E-05	0	0
25	6	0	0	0	-931.6114527	1277.587479
26	7	0	0	0	-3064.092798	8722.412521

各杆轴力展示：

表 3.2 各杆轴力输出表

轴编号	轴起点编号	轴终点编号	轴力大小/N
1	1	3	-5658.319723
2	1	4	-4333.281591
3	2	3	1317.497551
4	2	4	-2209.198932
5	3	4	1835.950576
6	3	5	-1959.146242
7	3	6	-3913.923756
8	4	5	1736.855386
9	4	6	-6501.433951
10	5	6	2270.492879
11	5	7	2767.63108
12	5	8	-4947.817209
13	6	7	702.9619333
14	6	8	-9766.06513
15	7	8	-3761.76938
16	7	9	4616.983343
17	8	9	-7064.639028
18	8	10	-1060.343279
19	8	11	-8768.21064
20	9	10	5373.924984
21	9	11	-535.2385676
22	10	11	1871.649564
23	10	12	8411.183881
24	10	13	-8020.837181
25	11	12	6121.298443
26	11	13	-11063.71141
27	12	13	-3018.199513
28	12	14	14049.80764

29	12	15	-1852.919757
30	13	14	-1852.919757
31	13	15	-15425.08764
32	14	15	-3018.199513
33	14	16	8411.183881
34	14	17	6121.298443
35	15	16	-8020.837181
36	15	17	-11063.71141
37	16	17	1871.649564
38	16	18	-1060.343279
39	16	19	5373.924984
40	17	18	-8768.21064
41	17	19	-535.2385676
42	18	19	-7064.639028
43	18	20	-3761.76938
44	18	21	-9766.06513
45	18	22	-4947.817209
46	19	20	4616.983343
47	20	21	702.9619333
48	20	22	2767.63108
49	21	22	2270.492879
50	21	23	-6501.433951
51	21	24	-3913.923756
52	22	23	1736.855386
53	22	24	-1959.146242
54	23	24	1835.950576
55	23	25	-2209.198932
56	23	26	-4333.281591
57	24	25	1317.497551
58	24	26	-5658.319723

4. 总结与展望

4.1 我们的优点

在最初完成作业任务的时候，团队内有过分歧：我们究竟是“就事论事”，把这道题目做出来就好呢，还是我们创造一条“通用流程”，用来批量处理该问题。后来在尝试着手开始编程的时候，我们组发现要同时处理 26 个节点 58 根杆的话，哪怕是变量命名都会出很大的问题，更别说对号入座可能导致的遗漏、缺损等。于是我们组改换思路，创造了一套通用的求解器，走上了“通用流程”这条思路，供使用者以较低的成本求解系统。

我们的优点大致来讲如下：

1. 核心求解代码全部自主开发，自主可控，可移植性、可拓展性强；
2. 适用于任意复杂的二维平面桁架问题；
3. 对使用者本身的编程能力没有要求，可以以接近自然语言的方式编写有限元代码；
4. 支持自动导出报告信息，以及计算图片；
5. 源码 myFEM 框架还在持续更新中，预计后续会支持三维、杆板薄壁结构和梁单元的代码支持。

在本组内，虽然只有一位同学有编程能力，但在其代码封装下，即使不会 python 语法的团队成员也能调用其封装好的函数和类型，从而完成有限元求解过程。我们在开发 myFEM 求解框架的过程中，由一位同学开发，多位同学反馈测试，迭代升级，现在已经支持了诸多功能；预计在后续过程中还会有更多的新功能增入。

比如在 myFEM 最初版本中，只支持基础的铰接点位移约束；但在组员的反馈下，更新支持了弹性支承约束。比如下述算例为组员展示的弹簧约束算例：

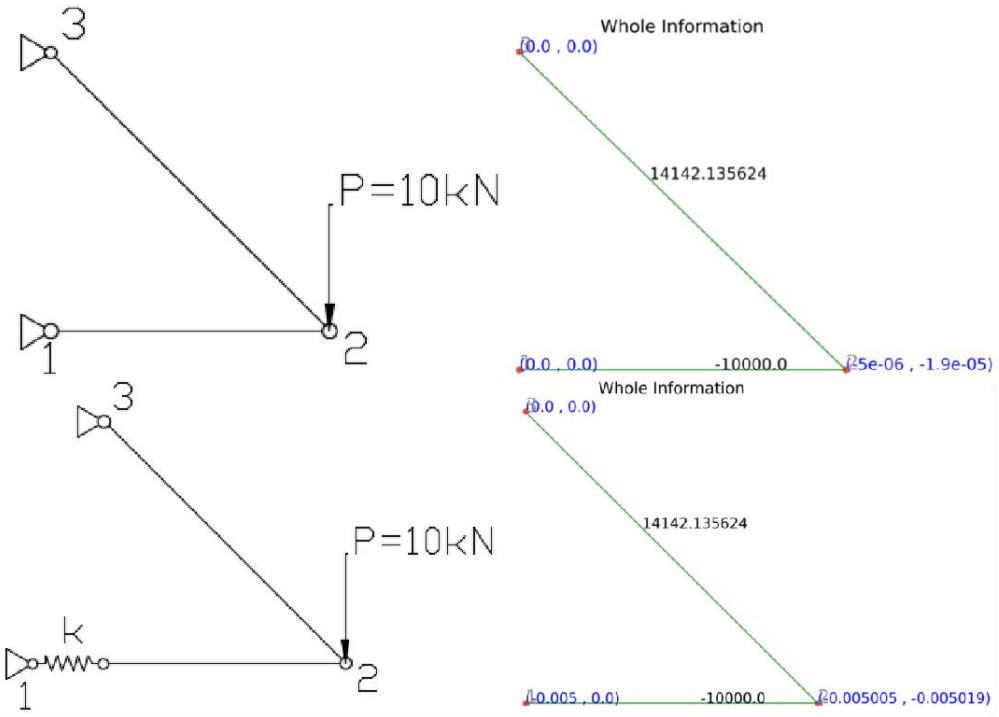


图 4.1 弹性支承对照算例

（弹簧约束参数：E = 200e9、f = 0.01、a = 1、k = 2e6）

我们会在 github 上持续发布我们的后续更新的源码及技术文档。

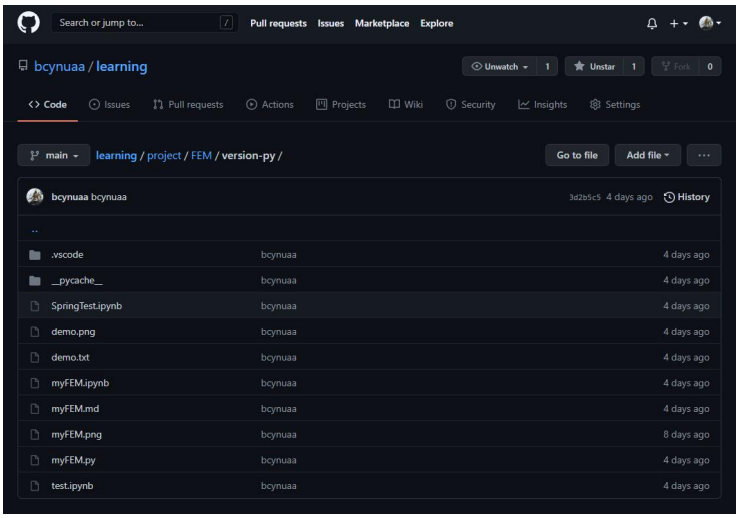


图 4.2 github 项目展示

4.2 我们的不足

本项目所开发的代码，仅仅适用于平面桁架的模型，各个杆在节点处铰接，节点处只能传递力而不能传递弯矩。这样，我们开发的代码不能适用于杆中存在弯矩的情况，而仅考虑了轴力和其引起的轴向应变。如果需要处理杆受弯矩而产生挠度的情况，还需要对本项目加以改进，用梁单元的模型来进行结构的离散化，进而求解挠度和弯矩等的数值。另外，由于本题目中没有出现受剪板单元，所以本项目尚不能解决受剪板以及相应的剪流计算问题，相比于其他队伍选择的题目中的杆板结构，我们仅有桁架结构，这也算是存在一个小小的缺陷了。简单罗列不足如下：

1. 暂不支持三维杆系结构；
2. 暂不支持薄壁结构；
3. 暂不支持梁单元；
4. 暂无薄板弯曲、平面问题的求解源码；
5. 暂时不支持置大数法、主元素置 1 法求解，仅支持分块矩阵求解。

不过正所谓大成若缺，有缺憾才会有进步，我们团队会在期末考试结束后持续更新有限元求解代码，争取能寻找到一个通用的、自主的求解架构。

4.3 展望

在过去的一年中，国际形势风雨飘摇，发生了很多事情；其中有一件对学术、工业界都有极大影响的，是哈尔滨工业大学被 MATHWORK 公司禁用 MATLAB 的事。

在我们团队中，所有人都是 MATLAB 爱好者；在之前的学习中，我们会使用它编写程序，仿真模拟，数据处理，绘图作图等等。但受上述事件影响，我们组最终决定使用开源的 python 语言。

一方面，是因为 python 有更高的灵活、开源特性，其不会像 MATLAB 一样被制裁、被禁用；另一方面，我们组想绕开 MATLAB 封装好的种种函数，自己从底层开始，让知识从代码中体现出来。自己独立自主地编写一套通用的求解框架。

自有限元大作业项目布置以来，过去了将近一周的时间，在完成这项大作业期间期间，我们小组遇到了很多难题，也收获很多成就。一步步走来，这其中的辛苦和考验只有经历过的人才知道。并且我们也经常就相关问题进行深入的探讨，我们也遇到了十分棘手的问题，有的时候甚至会萌生放弃的念头，这过程是十分艰辛的。幸好队友们都很优秀，我们互帮互助，相互体谅，攻坚克难，最终完成了源码编程和报告撰写。

独立自主会是将来永恒的话题，只有谙于底层，才能做出创新；只有控制自主，才能保障安全；只有不断地学习，不断地尝试，才能走出属于我们自己的道路。当今科技飞速发展，很多领域的科技应用已经趋于成熟和饱和，但与此同时，世界上还有无穷无尽的新鲜事物等待着我们去发现，新生科技闪着微弱的光，只有善于思考、勇于创新的人才能先人一步，成为新生领域的先驱者。所以身为大学生的我们，在大学期间不仅要学习知识，更要培养自己的创新思维。

5 参考文献

[1] 史治宇，丁锡洪主编.飞行器结构力学[M].北京：科学出版社.2018.
[2] 裴尧尧，肖衡林，马强，李丽华著.Python 与有限元 基于 Python 编程的有限元分析及应用扩展[M].北京：中国水利水电出版社.2017.

6 附件

包括：

1.myFEM 框架下问题求解源程序
2.myFEM 纯粹杆系结构有限元求解 python 程序源码
3.myFEM 纯粹杆系结构有限元求解 python 程序说明文档
4.问题求解生成报告
5.问题求解生成轴力图及节点位移图
6.信息记录表(包含轴力表以及节点信息表)
7.附件.rar 文件压缩包

下页起。

附件Appendix

附件1、myFEM框架下问题求解源程序

```
import numpy as np # numpy计算库
import matplotlib.pyplot as plt # matplotlib绘图库
import myFEM as fem # 导入我写的有限元方法计算库

P = 10e3 #外载荷
a = 100e-2 #边长
f = 20e-4 #横截面积
E = 7e6 * 10**4 #弹性模量
mu = 0.3 #泊松比，似乎没有用到

nodeList = list() #节点列表，预设为空

for i in range(4): #1-8节点加入
    tmp1 = fem.Node(0 , i * a , Px = 0 , Py = 0)
    tmp2 = fem.Node(a , i * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
    pass

nodeList.append( fem.Node(a , 4*a , Px = 0 , Py = 0) ) #9节点

for i in range(5): #10-19节点
    tmp1 = fem.Node( (2 + i) * a , 3 * a , Px = 0 , Py = 0)
    tmp2 = fem.Node( (2+i) * a , 4 * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
    pass

nodeList.append( fem.Node(7 * a , 3 * a , Px = 0 , Py = 0) ) #20节点

for i in range(3): #剩余节点
    tmp1 = fem.Node( 6 * a , (2 - i) * a , Px = 0 , Py = 0)
    tmp2 = fem.Node( 7*a , (2 - i) * a , Px = 0 , Py = 0)
    nodeList.append(tmp1)
    nodeList.append(tmp2)
    pass

sys = fem.SysPole(nodeList) #系统生成

for i in [1 , 2 , 25 , 26]: #支座的约束生成
    sys.setUV(i , u = 0 , v = 0)
    sys.setP(i , Px = np.NaN , Py = np.NaN)
    pass

for i in [13 , 15]: #给定的外载荷约束
    sys.setP(i , Px = 0 , Py = - P)
    pass

del i , tmp1 , tmp2
```

```
for i in [1 , 3 , 5 , 8 , 10 , 12 , 14 , 16]: # 连接 5 根杆节点
    sys.connect(i , i+2 , E , f)
    sys.connect(i , i+3 , E , f)
    sys.connect(i+1 , i+2 , E , f)
    sys.connect(i+1 , i+3 , E , f)
    sys.connect(i+2 , i+3 , E , f)
    pass

for i in [26 , 24]: # 连接 5 根杆节点
    sys.connect(i-2 , i , E , f)
    sys.connect(i-3 , i , E , f)
    sys.connect(i-2 , i-1 , E , f)
    sys.connect(i-3 , i-1 , E , f)
    sys.connect(i-3 , i-2 , E , f)
    pass

# 连接剩下的节点
sys.connect(7 , 9 , E , f)
sys.connect(8 , 9 , E , f)
sys.connect(19 , 20 , E , f)
sys.connect(18 , 20 ,E , f)
sys.connect(18 , 22 , E , f)
sys.connect(18 , 21 , E , f)
sys.connect(20 , 22 , E , f)
sys.connect(20 , 21 , E , f)

del i

# 系统生成
sys.generate()

sys.solve() # 系统求解
sys.getpic(ftsize = 15) # 得到系统图片
sys.export('hw.txt' , 'hw.png') # 导出报告以及图片

#导出信息报告
import xlwt

excel = xlwt.Workbook(encoding = 'utf-8')
sheet1 = excel.add_sheet('轴力信息表')
sheet2 = excel.add_sheet('节点信息表')

#轴力信息录入
sheet1.write(0 , 0 , '轴编号')
sheet1.write(0 , 1 , '轴起点编号')
sheet1.write(0 , 2 , '轴终点编号')
sheet1.write(0 , 3 , '轴力大小/N')

k = 1
for j in range(sys.N):
    for i in range(sys.N):
        if sys.NodeConnection[j][i] == 0:
            continue
            pass
        else:
            sheet1.write(k , 0 , k)
            sheet1.write(k , 1 , j+1)
            sheet1.write(k , 2 , i+1)
```

```
            sheet1.write(k , 3 , sys.Npole[j][i])
            k += 1
            pass
        pass
    pass

#节点信息录入
sheet2.write(0 , 0 , '节点编号')
sheet2.write(0 , 1 , '节点x坐标')
sheet2.write(0 , 2 , '节点y坐标')
sheet2.write(0 , 3 , '节点水平位移u')
sheet2.write(0 , 4 , '节点垂直位移v')
sheet2.write(0 , 5 , '节点水平受载Px')
sheet2.write(0 , 6 , '节点垂直受载Py')

for j in range(sys.N):
    node = sys.NodeList[j]
    sheet2.write(j+1 , 0 , j+1)
    sheet2.write(j+1 , 1 , node.x)
    sheet2.write(j+1 , 2 , node.y)
    sheet2.write(j+1 , 3 , node.u)
    sheet2.write(j+1 , 4 , node.v)
    sheet2.write(j+1 , 5 , node.Px)
    sheet2.write(j+1 , 6 , node.Py)
    pass

excel.save('hw.xlsx')
```

附件2、myFEM纯粹杆系结构有限元求解python程序源码

```
# modyfied by bcynuaa 2021/6/9


import numpy as np
import matplotlib.pyplot as plt
NaN = np.NaN

class Node:

    def __init__(self , x = 0 , y = 0 , Px = NaN , Py = NaN , u = NaN , v =
NaN):

        '''
        生成节点类型，需要给出x,y坐标
        Px，Py
        u，v
        约束可以不给出，默认为待求值
        '''

        self.x = x
        self.y = y
        self.setP(Px , Py)
        self.setUV(u , v)
        pass

    def setP(self , Px = NaN , Py = NaN):
```

```
        '''
        设置Px和Py约束，默认为未知量，待求
        '''

        self.Px = Px
        self.Py = Py
        pass

    def setUV(self , u = NaN , v = NaN):

        '''
        设置u和v约束，默认为未知量，待求
        '''

        self.u = u
        self.v = v
        pass

    pass




class SysPole:

    def __init__(self , ls):

        '''
        初始化函数，传入一个Node类列表，从而建立系统
        NodeList : 存放着节点顺序表
        N : 为节点个数
        NodeConnection : 为连接矩阵，有向，暂时为全0列
        dictK : 字典类型，用以存放各种名称的部分刚度矩阵
        dictConnection : 字典类型，用以存放某个连接的连接信息
        K : 组装刚度矩阵，暂时全为0
        springList : 弹性元器件列表，存储着弹性元器件存放列表，暂时为空
        report : 生成报告字符串
        '''

        self.NodeList = ls
        self.N = len(ls)
        self.NodeConnection = np.zeros([self.N , self.N])
        self.dictK = dict()
        self.dictConnection = dict()
        self.K = np.zeros([2 * self.N , 2 * self.N])
        self.springList = list()
        self.report = 'Report : Information of Each Node\n\n'
        print("successfully build system !")
        pass


    def __getK(self , E = 200e9 , A = 0.01 , l = 1):

        '''
        根据E，A，l给出杆单元的局部刚度矩阵
        '''
        mat = np.array([
```

```
            [1 , -1],
            [-1 , 1]
        ])
        mat = E * A / l * mat
        return mat
        pass

    def connect(self , j , i , E = 200e9 , A = 0.01):

        '''
        根据给定的刚度参数E，A来连接节点j , i，有向
        更新NodeConnection连接矩阵信息
        并且在dictK中附加键名为'j,i'的刚度矩阵键值
        '''

        self.NodeConnection[j-1][i-1] = 1
        node1 = self.NodeList[j-1]
        node2 = self.NodeList[i-1]
        x = node2.x - node1.x
        y = node2.y - node1.y
        l = np.sqrt(x**2 + y**2)
        lambda1 = x / l
        lambda2 = y / l
        dire = np.array([
            [lambda1 , lambda2 , 0 , 0],
            [0 , 0 , lambda1 , lambda2]
        ])
        k = self.__getK(E , A , l)
        K = dire.T.dot( k ).dot( dire )
        self.dictK[str(j) +  ',' + str(i)] = K
        connect = [E , A , l , dire]
        self.dictConnection[str(j) + ',' +str(i)] = connect
        pass


    def __addK(self , j , i):

        '''
        在总体刚度矩阵K中，组装上dictK['j,i']这一个刚度矩阵
        分开传入j和i即可
        '''

        Kji = self.dictK[str(j) + ',' + str(i)]
        kjj = Kji[0:2 , 0:2]
        kii = Kji[2:4 , 2:4]
        kji = Kji[0:2 , 2:4]
        kij = Kji[2:4 , 0:2]
        self.K[2*(j-1) : 2*j , 2*(j-1) : 2*j] += kjj
        self.K[2*(i-1) : 2*i , 2*(i-1) : 2*i] += kii
        self.K[2*(j-1) : 2*j , 2*(i-1) : 2*i] += kji
        self.K[2*(i-1) : 2*i , 2*(j-1) : 2*j] += kij
        pass


    def generate(self):

        '''
        connect结束后，进行系统生成
```

```
        会更新最终的组装刚度矩阵K
        '''

        for j in range(self.N):
            for i in range(self.N):
                if self.NodeConnection[j][i] == 0:
                    continue
                else:
                    self.__addK(j+1 , i+1)
                    pass
                pass
            pass
        print("successfully get matrix K !")
        pass


    def setP(self , k , Px = NaN , Py = NaN):

        '''
        设置k号节点上Px和Py约束参数，默认不设置，即为代求量
        '''

        self.NodeList[k-1].setP(Px , Py)
        pass


    def setUV(self , k , u = NaN , v = NaN):

        '''
        设置k号节点上u和v约束参数，默认不设置，即为代求量
        '''

        self.NodeList[k-1].setUV(u , v)
        pass

    def setSpring(self , i , k = 2e9 , dire = 'x'):
        tmp = dict()
        tmp['id'] = i-1
        tmp['k'] = k
        tmp['direction'] = dire
        if dire == 'x':
            self.NodeList[i-1].Px = 0
            self.NodeList[i-1].u = NaN
            self.K[2*(i-1)][2*(i-1)] += k
            pass
        else:
            self.NodeList[i-1].Py = 0
            self.NodeList[i-1].v = NaN
            self.K[2*i-1][2*i-1] += k
            pass
        self.springList.append(tmp)
        pass


    def __getpartK(self , row , col):

        '''
        得到刚度矩阵row和col所控制的切片矩阵
```

```
        返回刚度矩阵K，在row中序号、以及col序号所相交形成的矩阵
        '''

        m = row.size
        n = col.size
        mat = np.zeros([m , n])
        for j in range(m):
            for i in range(n):
                mat[j][i] = self.K[ int(row[j]) ][ int(col[i]) ]
                pass
            pass
        return mat
        pass


    def __modifyK(self , pNaN , uvNaN):

        '''
        将K刚度矩阵调整，分割为4块
        让已知约束力和已知约束位移对应在一起
        生成：m*n , m*m , n*n , n*m的四块分块矩阵并返回
        '''

        n = pNaN.size
        m = uvNaN.size
        k11 = self.__getpartK(uvNaN , pNaN)
        k12 = self.__getpartK(uvNaN , uvNaN)
        k21 = self.__getpartK(pNaN , pNaN)
        k22 = self.__getpartK(pNaN , uvNaN)
        return k11 , k12 , k21 , k22
        pass

    def __getSolve(self):

        '''
        用于solve的预处理
        分解还得代求量
        返回：
        p , uv , pNaN , uvNaN , p1 , p2 , uv1 , uv2
        '''

        P = np.zeros([self.N , 2])
        UV = np.zeros([self.N , 2])
        for i in range(self.N):
            node = self.NodeList[i]
            P[i] = np.array([node.Px , node.Py])
            UV[i] = np.array([node.u , node.v])
            pass
        #重塑为1*2*N矩阵
        p = P.flatten()
        uv = UV.flatten()

        #提取NaN位置信息，知道哪些未知量需要被处理
        pNaN = list()
        uvNaN = list()
        for j in range(2*self.N):
            if np.isnan(p[j]) == True:
                pNaN.append(j)
```

```
                pass
            if np.isnan(uv[j]) == True:
                uvNaN.append(j)
                pass
            pass
        pNaN = np.array(pNaN) #存储了力向量未知量的位置，也即位移向量已知量的位置
        uvNaN = np.array(uvNaN) #存储了位移向量未知量的位置，也即力向量已知量的位置

        p1 = list()
        p2 = list()
        uv1 = list()
        uv2 = list()
        for j in range(2*self.N):
            if j in pNaN:
                p2.append(p[j])
                pass
            elif j not in pNaN:
                p1.append(p[j])
                pass
            if j in uvNaN:
                uv2.append(uv[j])
                pass
            elif j not in uvNaN:
                uv1.append(uv[j])
                pass
            pass
        p1 = np.array(p1) #存储力已知量列向量
        p2 = np.array(p2) #存储力未知量列向量，NaN
        uv1 = np.array(uv1) #存储位移已知量列向量
        uv2 = np.array(uv2) #存储位移未知量列向量，NaN

        return p , uv , pNaN , uvNaN , p1 , p2 , uv1 , uv2
        pass


    def solve(self):

        '''
        在约束给定以后，求解整个系统情形
        方法为分块矩阵求解，调整对应已知量关系求解
        更新信息进入NodeList中的每一个节点
        并且在连接矩阵的索引下，生成内力矩阵
        '''
        p , uv , pNaN , uvNaN , p1 , p2 , uv1 , uv2 = self.__getSolve()

        #矩阵分块
        k11 , k12 , k21 , k22 = self.__modifyK(pNaN , uvNaN)

        #开始求解
        uv2 = np.linalg.inv(k12).dot(p1 - k11.dot(uv1))
        p2 = k21.dot(uv1) + k22.dot(uv2)
        t1 = 0
        t2 = 0
        for k in range(2*self.N):
            if np.isnan(p[k]) == True:
                p[k] = p2[t1]
                t1 += 1
                pass
```

```
            if np.isnan(uv[k]) == True:
                uv[k] = uv2[t2]
                t2 += 1
                pass
            pass
        #求解完毕，p与uv中已存储完毕整个力场与位移场

        #将p和uv写入各个节点
        for k in range(self.N):
            self.NodeList[k].setP(Px = p[2*k] , Py = p[2*k+1])
            self.NodeList[k].setUV(u = uv[2*k] , v = uv[2*k+1])
            pass
        #对弹性元器件作用的节点上，进行原先置0力的更新
        for spring in self.springList:
            i = spring['id']
            k = spring['k']
            if spring['direction'] == 'x':
                self.NodeList[i].Px = k * self.NodeList[i].u
                pass
            else:
                self.NodeList[i].Py = k * self.NodeList[i].v
                pass
            pass
        print("successfully update node infomation !")

        #生成各个节点杆力,存入Npole矩阵中
        self.Npole = np.zeros([self.N , self.N])
        for j in range(self.N):
            for i in range(self.N):
                if self.NodeConnection[j][i] == 0:
                    continue
                else:
                    connect = self.dictConnection[str(j+1) + ',' +str(i+1)]
                    e = connect[0]
                    a = connect[1]
                    l = connect[2]
                    ke = self.__getK(e , a , l)
                    lam = connect[3]
                    nodej = self.NodeList[j]
                    nodei = self.NodeList[i]
                    delta = np.array([
                        nodej.u , nodej.v , nodei.u , nodei.v
                    ])
                    s = ke.dot(lam).dot(delta)
                    self.Npole[j][i] = s[1]
                    pass
                pass
            pass

        #生成报告文本
        self.__getreport()
        self.getpic()
        pass


    def __getreport(self):

        '''
```

```
        将solve完的结果更新到report字符串中
        '''

        for i in range(40):
            self.report += '*'
            pass
        self.report += '\n'
        self.report += '\nConnection Matrix : \n'
        self.report += str(self.NodeConnection) + '\n'
        self.report += '\nInternel Force Matrix : \n'
        self.report += str(self.Npole) + '\n\n\n'
        for k in range(self.N):
            for i in range(30):
                self.report += '-'
                pass
            node = self.NodeList[k]
            self.report += '\n Node' + str(k+1) + ':\n Position : ( '
            self.report += str(node.x) + ' , ' + str(node.y) + ' )\n\n'
            self.report += 'Externel Load : \n'
            self.report += 'Horizontal Load Px = ' + str(node.Px) + '\n'
            self.report += 'Vertical Load Py = ' + str(node.Py) + '\n\n'
            self.report += 'Displacement : \n'
            self.report += 'Horizontal Displacement u = ' + str(node.u) + '\n'
            self.report += 'Vertical Displacement v = ' + str(node.v) +'\n\n'
            pass
        for i in range(40):
            self.report += '*'
            pass
        pass


    def getpic(self , precision = 6 , size = 40 , ftsize = 50):

        '''
        绘制整个图像存储入self.pic中
        precision : 控制显示精度
        size : 控制图片大小
        ftsize : 控制字体大小
        '''

        self.pic = plt.figure(facecolor = 'white' , figsize = (size , size) )
        for i in range(self.N):
            node = self.NodeList[i]
            px = node.x
            py = node.y
            plt.scatter(px , py , s = 40*size , alpha = 0.7 , color = 'r')
            plt.text(px , py , '(' + str(round(node.u , precision)) + ' , ' +
str(round(node.v , precision)) + ')' , fontsize = ftsize , color = 'b')
            plt.text(px , py , str(i+1) , fontsize = ftsize+30 , color = 'gray'
, alpha = 0.8)
            pass
        for j in range(self.N):
            for i in range(self.N):
                if self.NodeConnection[j][i] == 0:
                    continue
                else:
                    nodej = self.NodeList[j]
                    nodei = self.NodeList[i]
```

```
                    plt.plot([nodej.x , nodei.x] , [nodej.y , nodei.y] ,
linewidth = size/4 , alpha = 0.7 , color = 'g')
                    px = 0.4*nodej.x + 0.6*nodei.x
                    py = 0.4*nodej.y + 0.6*nodei.y
                    plt.text(px , py , str(self.Npole[j][i].round(precision)) ,
fontsize = ftsize , color = 'k')
                    pass
                pass
            pass
        plt.axis('off')
        plt.title('Whole Information' , fontsize = ftsize+10)
        pass


    def info(self):

        '''
        打印报告信息
        '''

        print(self.report)
        self.pic.show()

        pass


    def export(self , filename = 'report.txt' , picname = 'pic.png'):

        '''
        将报告写入filename文件中
        画图
        给出精度格式
        '''

        f = open(filename , mode = 'w' , encoding = 'utf-8')
        f.write(self.report)
        f.close()
        print("successfully write to file " + filename + ' !')

        self.pic.savefig(picname)

        pass


    pass
```

附件3、myFEM纯粹杆系结构有限元求解python程序说明文档

myFEM有限元杆系求解架构说明文档

1. Node 节点 类型

1.1. 变量

- x : 节点x坐标，默认为0
- y : 节点y坐标位置，默认为0
- u : 节点x方向位移，默认为np.NaN即待求量
- v : 节点y方向位移，默认为np.NaN即待求量
- Px : 节点x方向外载，默认为np.NaN即待求量
- Py : 节点y方向外载荷，默认为np.NaN即待求量

1.2. 函数

1.2.1. __init__(self , x = 0 , y = 0 , Px = NaN , Py = NaN , u = NaN , v = NaN) 初始化函数

一般而言，对于某个节点只需要给出x和y坐标即可；剩下的参数属于可选参数；建议使用如下命令创建节点：

```
import myFEM as fem

node = fem.Node(position_x , position_y)
```

若该点为支座点，则可以在创建的时候直接给定约束：

```
node_support = fem.Node(position_x , position_y , u = 0 , v = 0)
```

若该点为自由节点，则也可以在创建的时候直接给定约束：

```
node_free = fem.Node(position_x , postion_y , Px = 0 , Py = 0)
```

1.2.2. setP(self , Px = NaN , Py = NaN) 力约束函数

例如对某个节点node，将其设置外载约束：

- Px = 10kN
- Py = -10kN

则只需要：

```
node.setP(Px = 10e3 , Py = -10e3)
# 或者直接按给定顺序输入，如：
# node.setP(10e3 , -10e3)也可以
```

即可

1.2.3. setUV(self , u = NaN , v = NaN) 位移约束函数

例如对某个节点node，将其设置位移约束：

- u = 1e-4
- v = 2e-4

则只需要:

```
node.setUV( u = 1e-4 , v= 2e-4)
# 或者直接按给定顺序输入，如：
# node.setUV(1e-4 , 2e-4)也可以
```

即可

2. SysPole 杆系统 类型

2.1. 变量

- NodeList : list()列表类型，存放着一连串的节点，暗含标号
- N： int类型，节点总个数
- NodeConnection : numpy数组类型，由 1 和 0 构成；其 j 行 i 列为 1 ，则表示有 j 号节点指向 i 号节点的杆单元连接；其概念类似dijsktra和floyd算法中的有向邻接矩阵
- dictK : dict()字典类型，存放着局部刚度矩阵k_{ji}，调用时应按照dictK['j,i']的格式调用，即k_{ji} = dcitK['j,i']
- dictConnection : dict()字典类型，其中存放列表，每个列表['j,i']表示连接 j 和 i 节点的连接信息，包括[0] : 弹性模量E，[1] : 横截面积A， [2] : 长度l
- K : numpy数组类型，表示系统的整体刚度矩阵，但可能会被弹性元器件更新
- springList : list()列表类型，存放着一系列弹性元器件的字典，每个字典中都有['k']刚度系数，['direction']方向(为'x'或者'y')，以及['id']支撑在数字为'id'的节点上
- Npole： numpy数组类型，j 行 i 列上记录着 j 连接 i 的杆上轴力

- report : 字符串类型，为最终打印生成的报告
- pic : matplotlib图片类型，为最终根据计算结果绘制的轴力、节点位移图

2.2. 公有函数

2.2.1. __init__(self , ls) 初始化函数

初始化函数，传入一个全是Node的list列表，从而初始化系统，生成如下变量：

- NodeList : 存放着节点顺序表
- N : 为节点个数
- NodeConnection : 为连接矩阵，有向，暂时全为0
- dictK : 字典类型，用以存放各种名称的部分刚度矩阵，暂时为空
- dictConnection : 字典类型，用以存放某个连接的连接信息，暂时为空
- K : 组装刚度矩阵，暂时全为0
- springList : 弹性元器件列表，存储着弹性元器件存放列表，暂时为空
- report : 生成报告字符串，暂时只有标题

使用方法大致如下：

```
nodeList = [node1 , node2 , node3...nodeN]
system = fem.SysPole(nodeList)
```

运行结束后，会打印输出提醒:

```
successfully build system !
```

2.2.2. connect(self , j , i , E = 200e9 , A = 0.01) 连接节点函数

连接系统中的节点，需要对给定：

- j : 连接起点
- i : 连接终点
- E： 连接杆的弹性模量
- A： 横截面积

在这个函数中，会进行如下操作：

- 将NodeConnection的 j 行 i 列置为 1 ， 表明有 j 向 i 的杆连接
- 对dictConnection连接信息字典中的，['j,i']键值内添加列表[E,A,l]，其中l为杆长，根据节点相对位置计算而得
- 对dictK局部刚度矩阵，['j,i']键值设为计算而得的局部刚度矩阵，调用了__getK()私有函数

具体用法如下：

```
system.connect(j , i , E , A)
```

即完成了对节点间的杆连接

2.2.3. generate() 刚度矩阵生成函数

在连接结束系统之后，可以获取系统的整体刚度矩阵；把dictK中所有的局部刚度矩阵叠加到总体刚度矩阵K中，就获得了最终的刚度矩阵；过程中用到了__addK()私有函数

具体用法如下:

```
system.generate()
```

运行成功后，会打印输出提醒：

```
successfully get matrix K !
```

2.2.4. setP(self , k , Px = NaN , Py = NaN) 力约束函数

对 k 节点进行力约束，用法同Node类型中的setP()函数，只是现在需要指定节点编号 k

具体用法如下：

```
system.setP(i , Px = 1e4 , Py = -1e4)
```

2.2.5. setUV(self , k , u = NaN , v = NaN) 位移约束函数

用法同上，类似，不赘述

2.2.6. setSpring(self , i , k = 2e9 , dire = 'x') 弹性约束函数

对 i 节点的dire方向进行刚度系数为k的弹性约束，其会做如下操作：

- 在springList中，加入一个字典，该字典为['id':i , 'k':k , 'direction':dire]，记录了弹簧的连接信息
- 在 i 节点对应的dire方向上，在刚度矩阵主对角线上对应元素+k，并设置该节点约束为dire方向上外载荷为0；在后续求解过程中，求出的位移乘-k即获得该节点的弹性支承载荷

比如在3号节点的y方向，施加劲度系数为1e8的弹簧支座，则：

```
system.setSpring(3 , k = 1e8 , dire = 'y')
```

2.2.7. solve() 系统求解函数

在连接结束、约束给足、刚度矩阵生成之后，最终求解该矩阵；求解方式是分块矩阵求解；大致理论阐述如下：

位移和力的总共未知数量应该为节点数的两倍，已知数量等于未知数量——通过矩阵的分块求解，调整，可以得到如下的分块矩阵:

$$\begin{bmatrix} P_{1(m\times 1)} \\ P_{2(n\times 1)} \end{bmatrix} = \begin{bmatrix} A_{(m\times n)} & B_{(m\times m)} \\ C_{(n\times n)} & D_{(n\times m)} \end{bmatrix} \begin{bmatrix} uv_{1(n\times 1)} \\ uv_{2(m\times 1)} \end{bmatrix}$$

其中A, B, C, D, P_1, uv_1均已知，所以可以分块得到：

$$uv_2 = B^{-1}(P_1 - Auv_1)$$
$$P_2 = Cuv_1 + Duv_2$$

求解过程相当复杂，步骤大约如下：

- 调用__getSolve()私有函数，获取待求解的一系列值p , uv , pNaN , uvNaN , p1 , p2 , uv1 , uv2
- 调用__modifyK()私有函数，分块切片刚度矩阵获得k11 , k12 , k21 , k22
- 按照上述流程求解
- 将求解过程返还到NodeList中各个节点上，若有弹性支承力给定为-kx
- 更新Npole的各个轴力信息，获得杆的轴力矩阵
- 调用__getreport()私有函数，将报告存入report字符串
- 调用getpic()函数，绘制轴力图、节点位移图，更新pic变量

具体用法如下：

```
system.solve()
```

运行成功后，打印输出信息：

```
successfully update node infomation !
```

2.2.8. info(self) 信息打印函数

将report字符串打印到控制台上

具体用法如下：

```
system.info()
```

会在控制台(bash)中打印出report报告

2.2.9. export(self , filename = 'report.txt' , picname = 'pic.png') 导出计算结果函数

- 导出report至filename中
- 导出pic至picname中

具体用法如下：

```
system.export('demo.txt' , 'demo.png')
```

即在同目录下生成报告文件'dem.txt'和图片文件'demo.png'

2.2.10. getpic(self , precision = 6 , size = 40 , ftsize = 50) 绘图函数

更新pic变量，重绘

- precision 绘图标注的数据精度
- 图像大小
- 字体大小

具体用法如下：

```
system.getpic(precision = 8 , size = 50 , ftsize = 90)
```

2.3. 私有函数

因为私有函数是在类内部调用，所以这里从简说明

2.3.1. __getK(self , E = 200e9 , A = 0.01 , l = 1) 获取局部刚度矩阵函数

即：

$$\frac{EA}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

返回该矩阵

2.3.2. __addK(self , j , i) 组装矩阵函数

把dictK['j.i']加入K中，也即将k_{ji}加入总体刚度矩阵K中

2.3.3. __getpartK(self , row , col) 矩阵提取函数

将总体刚度矩阵K中的row行编号，和col列编号，相交获得的局部矩阵提取出来，并返回

2.3.4. __modifyK(self , pNaN , uvNaN) 矩阵分块函数

pNaN记录着未知力编号，uvNaN记录着位移未知编号，以此控制分块总体刚度矩阵，获得四块矩阵k11 , k12 , k21 , k22并返回

2.3.5. __getSolve(self) 求解预处理函数

获取：

- p : 力向量，包含代求值
- uv： 位移向量，包含代求值
- pNaN： 未知力编号
- uvNaN： 未知位移编号
- p1： 已知力向量
- p2： 未知力向量
- uv1： 已知位移向量
- uv2： 未知位移向量

2.3.6. __getreport(self) 生成报告字符串函数

把计算结果写入report字符串内，并存储

附件4、问题求解生成报告

```
Report : Information of Each Node

****************************************

Connection Matrix :
[[0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0.]
 [0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
 0. 0.]
[0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0.
 1. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 1. 1.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0.]]

Internel Force Matrix :
[[     0.              0.          -5658.31972307  -4333.2815908
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
 [     0.              0.           1317.49755126  -2209.19893197
       0.              0.              0.              0.
```

```
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.           1835.95057596
  -1959.14624173  -3913.92375562      0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
    1736.85538644  -6501.43395131      0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.           2270.49287872   2767.63108036  -4947.81720879
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.            702.96193327  -9766.06512989
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.          -3761.76938023
    4616.98334277      0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
   -7064.6390282   -1060.34327854  -8768.21063954      0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.           5373.92498419   -535.23856761      0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.           1871.64956365   8411.18388092
  -8020.83718111      0.              0.              0.
```

```
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.           6121.29844262
  -11063.71140815      0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
   -3018.19951342  14049.80764438  -1852.91975679      0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.          -1852.91975679 -15425.08764469      0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.          -3018.19951342   8411.18388092
    6121.29844262      0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.          -8020.83718111
  -11063.71140815      0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
    1871.64956365  -1060.34327854   5373.92498419      0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.          -8768.21063954   -535.23856761      0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.          -7064.6390282   -3761.76938023
   -9766.06512989  -4947.81720879      0.              0.
```

```
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.           4616.98334277
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
     702.96193327   2767.63108036      0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.           2270.49287872  -6501.43395131  -3913.92375562
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.           1736.85538644  -1959.14624173
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.           1835.95057596
   -2209.19893197  -4333.2815908 ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
    1317.49755126  -5658.31972307]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]
[      0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.              0.              0.
       0.              0.        ]]
```

```
------------------------------
 Node1:
 Position : ( 0 , 0.0 )

Externel Load :
Horizontal Load Px = 3064.092797647294
Vertical Load Py = 8722.41252072207

Displacement :
Horizontal Displacement u = 0.0
Vertical Displacement v = 0.0

------------------------------
 Node2:
 Position : ( 1.0 , 0.0 )

Externel Load :
Horizontal Load Px = 931.6114526916572
Vertical Load Py = 1277.587479277875

Displacement :
Horizontal Displacement u = 0.0
Vertical Displacement v = 0.0

------------------------------
 Node3:
 Position : ( 0 , 1.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -5.923796303994329e-05
Vertical Displacement v = -4.0416569450534104e-05

------------------------------
 Node4:
 Position : ( 1.0 , 1.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -4.6124030354539406e-05
Vertical Displacement v = -1.5779992371210942e-05

------------------------------
 Node5:
 Position : ( 0 , 2.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
```

```
Horizontal Displacement u = -0.00010956672896679891
Vertical Displacement v = -5.4410471177212084e-05

-------------------------------
 Node6:
 Position : ( 1.0 , 2.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -9.334892269025637e-05
Vertical Displacement v = -6.221880630912222e-05

-------------------------------
 Node7:
 Position : ( 0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -7.581410745963444e-05
Vertical Displacement v = -3.464167746051O7e-05

-------------------------------
 Node8:
 Position : ( 1.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -0.00010268388874697038
Vertical Displacement v = -0.000131976414379751

-------------------------------
 Node9:
 Position : ( 1.0 , 4.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 0.0001379392414150189
Vertical Displacement v = -0.00018243812172405506

-------------------------------
 Node10:
 Position : ( 2.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0
```

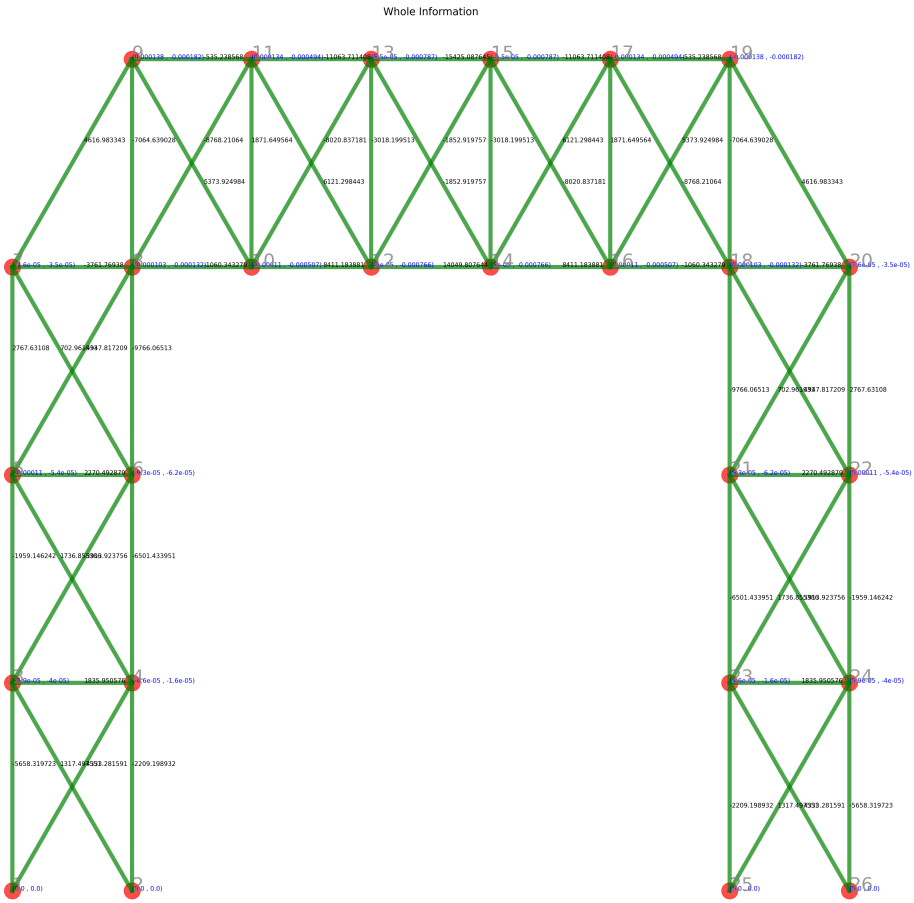
```
Displacement :
Horizontal Displacement u = -0.00011025776930798153
Vertical Displacement v = -0.0005074054893640399

------------------------------
 Node11:
 Position : ( 2.0 , 4.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 0.00013411610878921173
Vertical Displacement v = -0.0004940365639093902

------------------------------
 Node12:
 Position : ( 3.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -5.017788444423181e-05
Vertical Displacement v = -0.0007657776777516502

------------------------------
 Node13:
 Position : ( 3.0 , 4.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = -10000.0

Displacement :
Horizontal Displacement u = 5.5089598731023074e-05
Vertical Displacement v = -0.0007873362457046699

------------------------------
 Node14:
 Position : ( 4.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 5.0177884444230396e-05
Vertical Displacement v = -0.0007657776777516503

------------------------------
 Node15:
 Position : ( 4.0 , 4.0 )

Externel Load :
```

```
Horizontal Load Px = 0.0
Vertical Load Py = -10000.0

Displacement :
Horizontal Displacement u = -5.508959873102437e-05
Vertical Displacement v = -0.0007873362457046697

-------------------------------
 Node16:
 Position : ( 5.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 0.00011025776930798012
Vertical Displacement v = -0.00050740548936404

-------------------------------
 Node17:
 Position : ( 5.0 , 4.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -0.00013411610878921309
Vertical Displacement v = -0.0004940365639093904

-------------------------------
 Node18:
 Position : ( 6.0 , 3.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 0.00010268388874696905
Vertical Displacement v = -0.00013197641437975123

-------------------------------
 Node19:
 Position : ( 6.0 , 4.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = -0.00013793924141502O3
Vertical Displacement v = -0.00018243812172405522

-------------------------------
 Node20:
 Position : ( 7.0 , 3.0 )
```

```
Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 7.581410745963321e-05
Vertical Displacement v = -3.4641677746050866e-05

------------------------------
 Node21:
 Position : ( 6.0 , 2.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 9.334892269025565e-05
Vertical Displacement v = -6.221880630912246e-05

------------------------------
 Node22:
 Position : ( 7.0 , 2.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 0.00010956672896679821
Vertical Displacement v = -5.441047117721186e-05

------------------------------
 Node23:
 Position : ( 6.0 , 1.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 4.612403035453916e-05
Vertical Displacement v = -1.5779992371211094e-05

------------------------------
 Node24:
 Position : ( 7.0 , 1.0 )

Externel Load :
Horizontal Load Px = 0.0
Vertical Load Py = 0.0

Displacement :
Horizontal Displacement u = 5.923796303994305e-05
Vertical Displacement v = -4.0416569450533955e-05

------------------------------
```

```
 Node25:
 Position : ( 6.0 , 0.0 )

Externel Load :
Horizontal Load Px = -931.6114526916529
Vertical Load Py = 1277.5874792779005

Displacement :
Horizontal Displacement u = 0.0
Vertical Displacement v = 0.0

------------------------------
 Node26:
 Position : ( 7.0 , 0.0 )

Externel Load :
Horizontal Load Px = -3064.09279764729
Vertical Load Py = 8722.412520722044

Displacement :
Horizontal Displacement u = 0.0
Vertical Displacement v = 0.0

****************************************
```

附件5、问题求解生成轴力图及节点位移图



附件6、信息记录表

表1、轴力表

轴编号	轴起点编号	轴终点编号	轴力大小/N
1	1	3	-5658.319723
2	1	4	-4333.281591
3	2	3	1317.497551
4	2	4	-2209.198932
5	3	4	1835.950576
6	3	5	-1959.146242
7	3	6	-3913.923756
8	4	5	1736.855386
9	4	6	-6501.433951
10	5	6	2270.492879
11	5	7	2767.63108
12	5	8	-4947.817209
13	6	7	702.9619333
14	6	8	-9766.06513
15	7	8	-3761.76938
16	7	9	4616.983343
17	8	9	-7064.639028
18	8	10	-1060.343279
19	8	11	-8768.21064
20	9	10	5373.924984
21	9	11	-535.2385676
22	10	11	1871.649564
23	10	12	8411.183881
24	10	13	-8020.837181
25	11	12	6121.298443
26	11	13	-11063.71141
27	12	13	-3018.199513
28	12	14	14049.80764
29	12	15	-1852.919757
30	13	14	-1852.919757

轴编号	轴起点编号	轴终点编号	轴力大小/N
31	13	15	-15425.08764
32	14	15	-3018.199513
33	14	16	8411.183881
34	14	17	6121.298443
35	15	16	-8020.837181
36	15	17	-11063.71141
37	16	17	1871.649564
38	16	18	-1060.343279
39	16	19	5373.924984
40	17	18	-8768.21064
41	17	19	-535.2385676
42	18	19	-7064.639028
43	18	20	-3761.76938
44	18	21	-9766.06513
45	18	22	-4947.817209
46	19	20	4616.983343
47	20	21	702.9619333
48	20	22	2767.63108
49	21	22	2270.492879
50	21	23	-6501.433951
51	21	24	-3913.923756
52	22	23	1736.855386
53	22	24	-1959.146242
54	23	24	1835.950576
55	23	25	-2209.198932
56	23	26	-4333.281591
57	24	25	1317.497551
58	24	26	-5658.319723

表2、节点信息表

节点编号	节点x坐标	节点y坐标	节点水平位移u	节点垂直位移v	节点水平受载Px	节点垂直受载Py
1	0	0	0	0	3064.092798	8722.412521
2	1	0	0	0	931.6114527	1277.587479
3	0	1	-5.9238E-05	-4.04166E-05	0	0
4	1	1	-4.6124E-05	-1.578E-05	0	0
5	0	2	-0.000109567	-5.44105E-05	0	0
6	1	2	-9.33489E-05	-6.22188E-05	0	0
7	0	3	-7.58141E-05	-3.46417E-05	0	0
8	1	3	-0.000102684	-0.000131976	0	0
9	1	4	0.000137939	-0.000182438	0	0
10	2	3	-0.000110258	-0.000507405	0	0
11	2	4	0.000134116	-0.000494037	0	0
12	3	3	-5.01779E-05	-0.000765778	0	0
13	3	4	5.50896E-05	-0.000787336	0	-10000
14	4	3	5.01779E-05	-0.000765778	0	0
15	4	4	-5.50896E-05	-0.000787336	0	-10000
16	5	3	0.000110258	-0.000507405	0	0
17	5	4	-0.000134116	-0.000494037	0	0
18	6	3	0.000102684	-0.000131976	0	0
19	6	4	-0.000137939	-0.000182438	0	0
20	7	3	7.58141E-05	-3.46417E-05	0	0
21	6	2	9.33489E-05	-6.22188E-05	0	0
22	7	2	0.000109567	-5.44105E-05	0	0
23	6	1	4.6124E-05	-1.578E-05	0	0
24	7	1	5.9238E-05	-4.04166E-05	0	0
25	6	0	0	0	-931.6114527	1277.587479
26	7	0	0	0	-3064.092798	8722.412521