矩阵位移法计算刚架内力

程序设计说明书

水工42张玍 2014010330

目录

[1. 问题描述 1](#_Toc463035027)

[2. 程序设计思路 1](#_Toc463035028)

[2.1 读入原始数据 2](#_Toc463035029)

[2.1.1 选择数据存储方式 2](#_Toc463035030)

[2.1.2 数据格式及数据读入函数 3](#_Toc463035031)

[2.2 生成整体刚度矩阵 5](#_Toc463035032)

[2.2.1 生成一般杆件局部坐标系下的刚度矩阵 5](#_Toc463035033)

[2.2.2 生成坐标转换矩阵 5](#_Toc463035034)

[2.2.3 将局部矩阵定位到整体矩阵中 5](#_Toc463035035)

[2.2.4 生成整体刚度矩阵K 6](#_Toc463035036)

[2.3 计算整体坐标系下的等效结点荷载 7](#_Toc463035037)

[2.3.1 将局部荷载定位到整体荷载 7](#_Toc463035038)

[2.3.2 计算整体坐标系下的等效结点荷载P 7](#_Toc463035039)

[2.4 解基本方程求解结点位移 8](#_Toc463035040)

[2.5 求杆端内力 8](#_Toc463035041)

[2.6 绘制内力图 9](#_Toc463035042)

[2.6.1 绘制刚架及添加文字 9](#_Toc463035043)

[2.6.2 绘制基础 10](#_Toc463035044)

[2.6.4 绘制弯矩图 11](#_Toc463035045)

[2.6.5 图象的处理及保存 13](#_Toc463035046)

# 问题描述

编写程序，用矩阵位移法求解刚架所有杆件的内力，并绘制内力图

# 程序设计思路

本程序既要计算，又要绘图，我选择了Python3。Python的强大并不在于它本身，而是在于它有着海量的第三方模块。本程序采用了numpy和matplotlib两个模块，分别可以进行科学计算和绘图。此外，还有导入了实现其他基本功能的模块。代码如下：

1. import numpy as np#导入numpy模块并命名为np
2. import numpy.linalg as nplg#导入numpy模块的linalg子模块并命名为nplg
3. import matplotlib.pyplot as plt#导入matplotlib模块的pyplot子模块并命名为plt
4. import os#导入操作系统函数模块
5. import win32ui#导入win32api的win32ui模块
6. import re#导入正则表达式模块

计算步骤参考课本P328，在此基础上我进行了一些改进，更便于计算机执行。以下分别说明。

## 2.1 读入原始数据

### 2.1.1 选择数据存储方式

读入数据之前先要选择数据的合适的储存方式。我认为本程序需要存储的是结点和单元，它们的特点是：每个结点或单元对应唯一一个编号；每个结点或单元有着自己的特性，因此可以用字典（dict）来存储它们。其中，编号为键值，其各种特性是值。对于结点而言，它的特性是自己的坐标，可以用一个二维数组储存（需要说明的是，Python中变量并不占据实际的内存空间，只是指向某一个类型的对象。但是为避免更多解释，以后均称为存储到某个变量中）。代码如下：

1. node = {}
2. Point = [x, y]
3. node[num] = Point

对于单元而言，它的特性相当之多，包括有：始端结点、末端结点、定位向量、EA值、EI值、长度l（本程序所有单元只对应杆件）、角度（整体坐标系顺时针旋转到局部坐标系为正）、均布荷载、刚度矩阵（局部坐标系下）、刚度矩阵（整体坐标系下）、单元固端约束力向量（局部坐标系下）、单元固端约束力向量（整体坐标系下）、杆端内力（局部坐标系下）、杆端内力（整体坐标系下）、单元等效结点荷载（整体坐标系下）、坐标转换矩阵、结点位移分量（局部坐标系下）、结点位移分量（整体坐标系下）。考虑到单元的特性如此之多，将其存储在一个类内，则类作为字典的值。代码如下：

此外，还定义了其他几个变量，包括存储结点位移数的shift、存储基础结点的ground、存储结点外荷载的P\_node，代码如下：

1. class Unit:
2. num = 0#单元编号
3. beginnode=[]#始端
4. endnode=[]#末端
5. vector=[]#定位向量
6. EA = 0
7. EI = 0
8. l = 0#长度
9. alpha = 0#角度α（整体坐标系顺时针旋转到局部坐标系为正）
10. q = 0#均布荷载
11. \_k = np.zeros((6,6))#刚度矩阵（局部坐标系下）
12. k = np.zeros((6,6))#刚度矩阵（整体坐标系下）
13. \_Fp = np.zeros(6)#单元固端约束力向量（局部坐标系下）
14. Fp = np.zeros(6)#单元固端约束力向量（整体坐标系下）
15. \_F = np.zeros(6)#杆端内力（局部坐标系下）
16. F = np.zeros(6)#杆端内力（整体坐标系下）
17. P = np.zeros(6)#单元等效结点荷载（整体坐标系下）
18. T = np.zeros((6,6))#坐标转换矩阵
19. delta = np.zeros(6)#结点位移分量（局部坐标系下）
20. \_delta = np.zeros(6)#结点位移分量（整体坐标系下）
21. unit = {}
22. shift = []#结点位移数
23. ground = []#基础
24. P\_node = []#结点外荷载
25. filename = []#文件名

这里需要说明的是，shfit作为结点位移数，只需要存储一个整数，而为何它和ground、P\_node这些可能需要存储多个数的变量一样也初始化为数组呢？这是因为shift需要在后面被读入（很惭愧，本程序并不能自动判断结点位移数）。而整数是不可更改的对象，在Python的函数中进行的只是变量传值操作，并不能改变整数的值。因此需要将shfit定义为一种可更改的对象，而list（数组）是可变的。filename也是如此。另外，由于在程序中不可能将局部坐标系的头部的一横打印出来，只得将这一横移到前面。当然不能是“-”，否则成了负号，而应该是“\_”。

### 2.1.2 数据格式及数据读入函数

只有满足一定格式的数据才能被读入并存储。我给出的example1如下：

1. 结点位移数,7
2. 结点,1,0,5
3. 结点,2,5,5
4. 结点,3,5,5
5. 结点,4,5,0
6. 结点,5,0,0
7. 基础,5,4
8. 单元,1,1,2,1,2,3,4,5,6,1000000,10000,5,0,10
9. 单元,2,1,5,1,2,3,0,0,0,100000,15000,5,90,0
10. 单元,3,3,4,4,5,7,0,0,0,100000,15000,5,90,0
11. 局部荷载,1,0,-25,-20.833,0,-25,20.833
12. 结点荷载,0,-30,0,0,0,0,0

每一行为中文的指令和具体的数据。中文指令包括：“结点位移数”、“结点”、“基础”、“单元”、“局部荷载”、“结点荷载”，每个指令可以将后面的数据存储到对应的变量中。本程序中，定义了一个函数“ReadData”来实现读入数据的功能。代码如下：

1. def ReadData(shfit,node,unit,P\_node):
2. dlg = win32ui.CreateFileDialog(1) # 1表示打开文件对话框
3. dlg.SetOFNInitialDir(os.getcwd()) # 设置打开文件对话框中的初始显示目录
4. dlg.DoModal() # 打开文件对话框
5. filename.append(dlg.GetPathName()) # 获取选择的文件名称
6. data = open(filename[0],'r')
7. for line in data:
8. text = line.split(',')
9. if text[0] == "结点位移数":
10. shfit.append(int(text[1]))
11. if text[0] == "结点":
12. num = int(text[1])
13. node[num] = list(map(lambda x: float(x), text[2:4]))
14. if text[0] == "基础":
15. ground.append(int(text[1]))
16. ground.append(int(text[2]))
17. if text[0] == "单元":
18. num = int(text[1])
19. unit[num] = Unit()
20. unit[num].num = num
21. unit[num].beginnode = node[int(text[2])]
22. unit[num].endnode = node[int(text[3])]
23. unit[num].vector = list(map(lambda x: int(x), text[4:10]))
24. unit[num].EA = float(text[10])
25. unit[num].EI = float(text[11])
26. unit[num].l = float(text[12])
27. unit[num].alpha = (float(text[13]) / 180) \* np.pi
28. unit[num].q = (float(text[14]))
29. if text[0] == "局部荷载":
30. num = int(text[1])
31. unit[num].\_Fp = list(map(lambda x: float(x), text[2:8]))
32. if text[0] == "结点荷载":
33. for x in text[1:]:
34. P\_node.append(float(x))
35. print("读入数据成功")

首先，程序将会弹出一个文件对话框，让用户选择文件，初始显示目录为当前项目路径。选择之后则按行依次读入数据。以“结点位移数,7”为例，读入这一行时，先以逗号为分割符分割字符串。分割后的第一个字符串为“结点位移数”，在指令范围中，则执行“结点位移数”指令,将第二个字符串“7”转为整数型并添加到结点位移shift后。此外，“单元”后依次是单元编号，始端编号，末端编号，单元定位向量，EA，EI，l，角度alpha，均布荷载q。注意单元定位向量应为六维向量，同时各个数之间必须用英文的逗号隔开。

## 生成整体刚度矩阵

### 2.2.1 生成一般杆件局部坐标系下的刚度矩阵

按照课本P305的公式，对于一般的杆件，已知EA、EI、l时可以计算出其刚度矩阵。为避免麻烦，我编写了一个自动生成刚度矩阵的函数K0，代码如下：

1. def K0(EA,EI,L):#本函数用于生成局部坐标系下的单元刚度矩阵
2. a = EA / L
3. b = EI / (L \*\* 1)
4. c = EI / (L \*\* 2)
5. d = EI / (L \*\* 3)
6. return np.array([
7. [a,0,0,-a,0,0],
8. [0,12\*d,6\*c,0,-12\*d,6\*c],
9. [0,6\*c,4\*b,0,-6\*b,2\*b],
10. [-a,0,0,a,0,0],
11. [0,-12\*d,-6\*c,0,12\*d,-6\*c],
12. [0,6\*c,2\*b,0,-6\*c,4\*b]
13. ])

### 2.2.2 生成坐标转换矩阵

按照课本P309的公式，对于一般的杆件，可以通过由整体坐标系旋转到局部坐标系的旋转角α计算出坐标转换矩阵。同样为避免麻烦，我编写了一个自动生成坐标转换矩阵的函数T0，代码如下：

1. def T0(alpha):#本函数用于通过旋转角α(alpha)生成单元坐标转换矩阵
2. return np.array([
3. [np.cos(alpha),np.sin(alpha),0,0,0,0],
4. [-np.sin(alpha), np.cos(alpha), 0, 0, 0, 0],
5. [0, 0, 1, 0, 0, 0],
6. [0,0,0,np.cos(alpha), np.sin(alpha), 0],
7. [0,0,0,-np.sin(alpha), np.cos(alpha), 0],
8. [0,0,0,0, 0, 1],
9. ])

### 2.2.3 将局部矩阵定位到整体矩阵中

假设局部码i,j分别对应总码、，则在局部坐标系下的单元刚度矩阵（简称局部矩阵）中排在第i行第j列的元素在整体刚度矩阵中排在第行第列。依据此原理，设计如下程序：

1. def P2W(K,k,lamda):#Part to Whole，局部矩阵定位到整体矩阵中
2. #K为整体阶段矩阵，k为当前局部矩阵(整体坐标系下)，lamda为当前定位向量
3. order = k.shape[0] #order为局部矩阵的阶数
4. nzlamda = [] # nonzero lamda，定位向量中的非零项
5. dict = {}#建立一个字典，字典的键值为定位向量中的非零项（即nzlamda中的元素），值为局部矩阵中对应该键值的数
6. #提取定位向量中的非零项并与原矩阵对应编码建立字典
7. for i in range(order):
8. if lamda[i] != 0:
9. dict[lamda[i]] = i + 1
10. nzlamda.append(lamda[i])
11. #将局部矩阵的第i行第j列对应到整体矩阵的第λi行第λj列中
12. for i in nzlamda:
13. for j in nzlamda:
14. i0 = dict[i]
15. j0 = dict[j]
16. K[i-1][j-1] += k[i0-1][j0-1]#这里各个项都要减1是因为计算机是从0开始存储的
17. return K

需要说明的是：本程序并未直接对局部矩阵进行搜索，再去通过定位向量定位到整体矩阵中。而是先找出能够定位到整体矩阵的局部码，也即对应总码不为0的局部码，也即定位向量中的非零项。由此可以减少信息的冗余度。同时，再将此非零项反对应到局部码，存到一个字典里（两者是一一对应的）。那么任意两个不同非零项的排列组合则对应同样数量的局部码的排列组合。这些组合即构成了所有可以对应到整体矩阵的局部矩阵的行列号，那么只需简单地将它们加到整体矩阵中就好。这样一来，虽然计算方式比直接搜索复杂多了，但是复杂度大大降低（复杂和复杂度是两个概念）。假如需要处理的局部矩阵是n阶的，则无论定位向量为何，直接搜索的复杂度都为O(n2),而本方法最多为O(n2)，最少为O(1)，具体不去细算。

### 2.2.4 生成整体刚度矩阵K

代码如下：

1. num\_shift = shift[0]
2. K = np.zeros((num\_shift,num\_shift))#整体刚度矩阵初始化
3. unit = UnitInput(unit)
4. for u in unit.values():#对于unit的所有键值，也即所有的单元
5. u.\_k = K0(u.EA,u.EI,u.l)
6. u.T = T0(u.alpha)
7. u.k = np.dot(np.dot(u.T.T,u.\_k),u.T)
8. K = P2W(K,u.k,u.vector)

整体刚度矩阵的阶数应当与结点位移数相同。整体坐标系下的单元刚度矩阵与局部坐标系下的单元刚度矩阵计算公式如下：



一次循环后即可生成整体刚度矩阵。

## 2.3 计算整体坐标系下的等效结点荷载

### 2.3.1 将局部荷载定位到整体荷载

全称为将局部坐标系下的单元等效结点荷载定位到整体坐标系下的整体结构的等效结点荷载中。原理类似局部矩阵定位到整体矩阵中，不再赘述。代码如下：

1. def P2W\_Vector(P,p,lamda):#局部等效结点荷载定位到整体荷载
2. length = len(p)#获得p的长度
3. nzlamda = [] # nonzero lamda，定位向量中的非零项
4. dict = {} # 建立一个字典，字典的键值为定位向量中的非零项（即nzlamda中的元素），值为局部矩阵中对应该键值的数
5. # 提取定位向量中的非零项并与原矩阵对应编码建立字典
6. for i in range(length):
7. if lamda[i] != 0:
8. dict[lamda[i]] = i + 1
9. nzlamda.append(lamda[i])
10. for i in nzlamda:
11. i0 = dict[i]
12. P[i-1] += p[i0 - 1] # 这里各个项都要减1是因为计算机是从0开始存储的
13. return P

代码也基本复制了之前的，也不赘述。

### 2.3.2 计算整体坐标系下的等效结点荷载P

代码如下：

1. P = np.zeros(num\_shift)#整体结点荷载初始化
2. for u in unit.values():
3. u.Fp = np.dot(u.T.T,u.\_Fp)
4. u.P = -u.Fp
5. P = P2W\_Vector(P,u.P,u.vector)
6. P = P + P\_node#注意加上结点外荷载

## 2.4 解基本方程求解结点位移

通过之前的步骤得到了整体刚度矩阵K和整体坐标系下的整体结构的等效结点荷载P。基本方程为：



则：



代码如下：

1. delta = np.dot(nplg.inv(K),P)#解基本方程得结点位移向量△(delta)

显然其中的nplg.inv()函数是用来求逆矩阵的。

## 2.5 求杆端内力

先求局部坐标系下的单元结点位移（简称局部结点位移），需要将整体的结点位移通过定位向量反定位到单元的结点位移中。通过函数W2P实现此功能，代码如下：

1. def W2P(D,d,lamda):#整体坐标系下，整体结点位移D反向定位到局部结点位移d
2. length = len(d) # 获得d的长度
3. nzle = [] # nonzero local encoing，对应的总码不为零的局部码
4. dict = {} # 建立一个字典，字典的键值为对应的总码不为零的局部码（即nz中的元素），值为对应的总码
5. # 提取定位向量中的非零项并与原矩阵对应编码建立字典
6. for i in range(length):
7. if lamda[i] != 0:
8. dict[i] = lamda[i]
9. nzle.append(i + 1)
10. for i in nzle:
11. i0 = dict[i - 1]
12. d[i - 1] += D[i0 - 1] # 这里各个项都要减1是因为计算机是从0开始存储的
13. return d

实现过程与之前的两个定位函数并无太大差别。之后，根据



计算出局部坐标系下的杆端内力。代码如下：

1. for u in unit.values():
2. u.delta = W2P(delta,u.delta,u.vector)
3. u.\_delta = np.dot(u.T,u.delta)
4. u.\_F = np.dot(u.\_k,u.\_delta) + u.\_Fp

## 2.6 绘制内力图

### 2.6.1 绘制刚架及添加文字

绘制刚架只需绘制各个单元，而各个单元只需连接两个结点，因此先定义一个line函数实现连线功能。代码如下：

1. def line(Point1,Point2,width=1):#连接端点为Point1与Point2的线段，宽度为width，默认为1
2. if Point2[0] != Point1[0]:
3. k = (Point2[1] - Point1[1]) / ((Point2[0] - Point1[0]) \* 1.0)
4. for x in np.linspace(Point1[0],Point2[0],200):
5. plt.scatter(x,k \* (x - Point1[0]) + Point1[1], color='k',s=width,marker='o',label=str)
6. elif Point2[0] == Point1[0]:
7. m = (Point2[0] - Point1[0]) / ((Point2[1] - Point1[1]) \* 1.0)
8. for y in np.linspace(Point1[1], Point2[1], 200):
9. plt.scatter( m \* (y-Point1[1]) + Point1[0],y, color='k',s=width, marker='o', label=str)

原理比较简单，根据两点公式求出直线方程，再可将画图区间细分，不断描点，点数足够多时在图上就呈现出直线。当然，点数越多图象越好，但同样会过多的点数会使得程序运行速度变慢。通过不断调整，我选择了分出200个点，结果比较漂亮的同时运行速度也较快。另外此函数还可定义直线宽度，默认值为1.

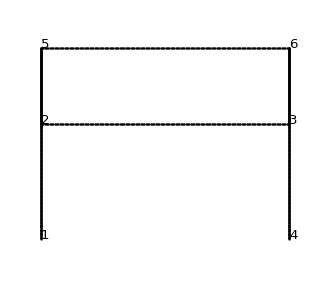
定义完连线函数之后，绘制刚架就简单了。代码如下：

1. for u in unit.values():
2. line(u.beginnode,u.endnode)

至于添加文字，为了避免结果太乱，只是添加了各个点号，代码如下：

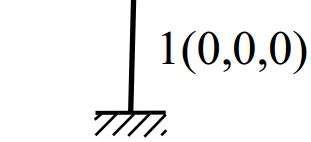
1. for k in node.keys():
2. n = node[k]
3. plt.text(n[0], n[1], str(k), color='k')

对于example2的绘制结果如图：



### 2.6.2 绘制基础

在给出的示例中，基础如图：



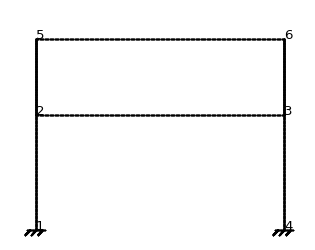
方法是用斜线填充矩形，矩形上部宽的中点为结点，矩形的另外三条边不绘制。这实现起来比较复杂，我的做法是先画出地面线，在地面线上选出几个点画一条较短的斜线。代码如下：

1. def base(Point):#画出基础
2. for x in np.linspace(Point[0] - 0.3, Point[0] + 0.3, 100):
3. plt.scatter(x, Point[1], color='k', s=0.1, marker='o', label=str)
4. for t in np.linspace(Point[0] - 0.2, Point[0] + 0.2, 3):
5. P1 = [t, Point[1]]
6. P2 = [t - 0.15, Point[1] - 0.25]
7. line(P1, P2, width=0.1)

绘制的基础如图：



虽然并不是很漂亮，但是勉强接受吧。加上基础后，对于example2的绘制结构如图

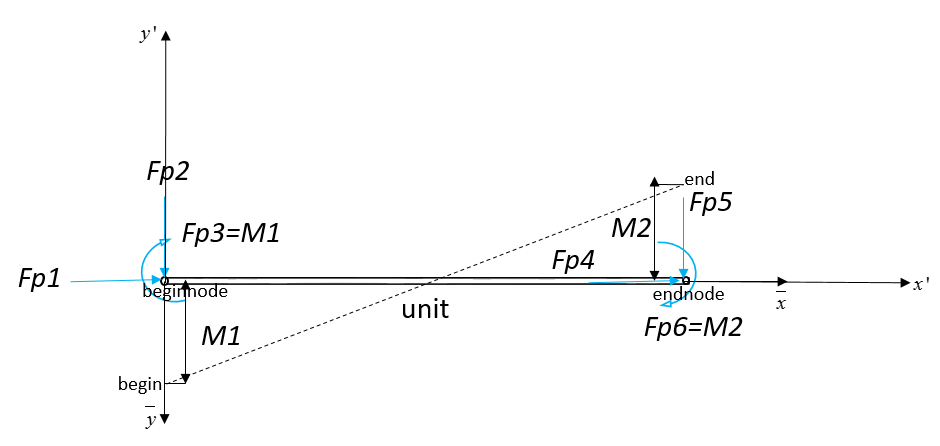


### 2.6.4 绘制弯矩图

实现此功能的函数为BMD（Bending Moment Diagram），代码如下：

1. def BMD(u,q=0, scanfactor=1 ):#Bending Moment Diagram弯矩图
2. #先在局部坐标系下求出M的方程
3. Fp = u.\_F#注意应为局部坐标系下的局部荷载！
4. alpha = u.alpha
5. L = u.l
6. begin = [0,-Fp[2]] #弯矩图的起点
7. end = [L,Fp[5]] #弯矩图的终点
8. k = (end[1] - begin[1]) /((end[0] - begin[0]) \* 1.0)
9. t = np.array([
10. [np.cos(alpha), -np.sin(alpha)],
11. [np.sin(alpha), np.cos(alpha)]
12. ])
13. for x in np.linspace(0,L,200):
14. y = (k \* (x - begin[0]) + begin[1] + q / 2.0 \* x \* (x- L)) / scanfactor
15. draw = np.dot(t,[x,y]) + u.beginnode
16. print(x, y)
17. plt.scatter(draw[0], draw[1], color='k', s=0.5, marker='o', label=str)
18. for x in np.linspace(0, L, 10):
19. y = (k \* (x - begin[0]) + begin[1] + q / 2.0 \* x \* (x- L)) / scanfactor
20. p1 = np.dot(t,[x, 0]) + u.beginnode
21. p2 = np.dot(t,[x, y]) + u.beginnode
22. line(p1,p2)

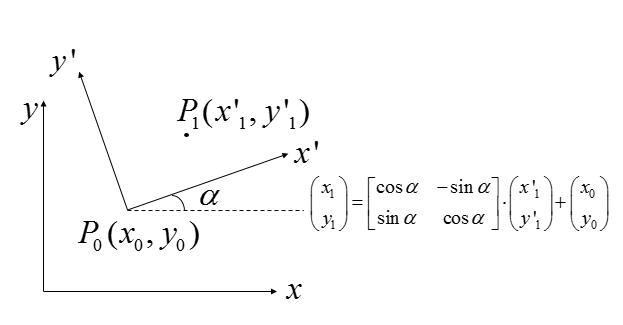
原理稍复杂。首先，我是在局部坐标系下求出弯矩图的方程，这一步比较简单。弯矩图应有两个端点，也即杆件两端的弯矩。解释图如下：



通过弯矩图的两个端点（begin和end）可以知道弯矩图的底图（直线）。再如果有均布荷载时，叠加即可。另外在函数中还定义了一个scanfactor(放缩因数)。这是因为当弯矩过大时可能超出绘图范围，因此需要进行一定的放缩（因为弯矩和杆长的单位是不统一的，自然可以放缩）。求放缩因数的代码如下：

1. F\_max = 0
2. for u in unit.values():
3. F\_max = max(F\_max, abs(u.\_F[2] / u.l),abs(u.\_F[5] / u.l))
4. scanfactor = F\_max \* 2

比较所有的弯矩绝对值除以杆长的商，得出最大值，则放缩因数是最大值的两倍。这样一来，如果对于超出杆长太多的弯矩，放缩以后也之后杆长的一半。通过以上的步骤，可以得出局部坐标系下的弯矩图的方程，且以杆件的始端为零点，杆件的方向作为坐标轴正向，杆件的方向的反方向作为坐标轴的正向。接下来将坐标转换到画图的整体坐标中（注意这个坐标与整体坐标不同）。解释图如下：





因此可以得到弯矩图在绘图坐标系中的坐标，也就能将它画出来了。至于弯矩图的填充，只要从杆件中选出几个点与弯矩图连接，而弯矩图的已知，坐标转换也已知。

BMD函数完成之后，绘制弯矩图就很简单了，代码如下：

1. for u in unit.values():
2. BMD(u,u.q,scanfactor)

### 2.6.5 计算结果的处理及保存

计算结果包括单元的各项数据及绘制的内力图。对于数据，保存函数如下：

1. def SaveData(unit):
2. print(">>>保存计算结果")
3. data = open(filename[0].split('.')[0]+'计算结果.txt','w')
4. for u in unit.values():
5. data.write('单元编号：'+str(u.num)+'\n')
6. data.write('局部坐标系下单元'+str(u.num)+'刚度矩阵：\n' + str(u.\_k) + '\n')
7. data.write('整体坐标系下单元'+str(u.num)+'刚度矩阵：\n' + str(u.k) + '\n')
8. data.write('局部坐标系下单元'+str(u.num)+'杆端内力：\n' + str(u.\_F)+'\n')
9. data.write('整体坐标系下单元'+str(u.num)+'杆端内力：\n' + str(u.F) + '\n')
10. data.write('局部坐标系下单元'+str(u.num)+'结点位移分量：\n' + str(u.\_delta) + '\n')
11. data.write('整体坐标系下单元'+str(u.num)+'结点位移分量：\n' + str(u.delta) + '\n\n')

对于图象，首先需要定出图幅大小，我认为只要能充分放入横纵坐标的最大值就可以了，代码如下：

1. x\_max = 0
2. y\_max = 0
3. for n in node.values():
4. x\_max = max(x\_max, n[0])
5. y\_max = max(y\_max, n[1])
6. plt.xlim(-x\_max/2, x\_max \* 1.5)
7. plt.ylim(-y\_max/2, y\_max \* 1.5)
8. node = {}
9. Point = [x, y]
10. node[num] = Point

接下来是添加图象标题、保存图象、显示图象：

1. #显示中文时由于matplotlib.pyplot在显示时无法找到合适的字体，会显示乱码（我的显示为方框），因此需要设定字体
2. examplename = re.split(r'\\|\.',filename[0])[-2]
3. plt.title(examplename+'弯矩图',fontproperties='SimHei')
4. plt.savefig(examplename+"弯矩图.png",dpi=200 )
5. plt.show()

本处的examplename使用了正则表达式对字符串进行分割。之前通过文件对话框的方式获得了文件路径，比如我的某一个路径如下：

D:\大三上课程\结构力学\Bigwork\example1.txt

可以看出，刚架算例的名称是example1，因此需要从整个字符串中将它分离出来，方法就是用正则表达式。r'\\|\.'的意思是用斜杠\或者点.进行分割，这样可以得到一个list，我们只要它的倒数第二项即可，即list[-2]。得到算例名后，图象的标题、保存的文件就不至于与其他的算例混淆了。

对于example2，保存的图象如下：

