计算物理第一次大作业

吴嘉晟　1900011382

2021 年 4 月 23 日

目录

自定义矩阵类

进行矩阵的定义和运算等操作时，可以使用 numpy 库产生的二维数组来实现。但本着**每一行代码都由自己编写**的原则，我使用嵌套列表自己实现了矩阵类 (class Matrix) 的接口，在之后的程序中所有的矩阵运算都通过自定义的矩阵类完成。

创建一个矩阵类可以有两种方式，一是给定一个的行数，列数和元素值，创建该维度的所有矩阵元都为的矩阵；二是直接用矩阵的每行形成的嵌套列表作为输入，创建矩阵。在不同情况下，这两种创建方式各有优劣。对于某些特殊矩阵（如基本矩阵，对角矩阵）或规模较大的稀疏矩阵，前者往往更具优势；对于已知各矩阵元的无规律矩阵或规模较小的矩阵，使用后者创建更加方便。其实现代码如下：

```
class Matrix:   # 矩阵类
    def __init__(self, value):
        if  isinstance(value,  tuple):        # 已知维度，创建矩阵
            self.size = (value[0], value[1])      # 矩阵的大小
            self.row = value[0]             # 矩阵的行数
            self.col = value[1]             # 矩阵的列数
            if  len(value) > 2:          # 矩阵所有元素相同
                self.matrix = [[value[2] for _ in  range(value[1])] for _ in  range(value[0])]
            else:
                self.matrix = [[0 for _ in  range(value[1])] for _ in  range(value[0])]     # 默认创建零矩阵
        elif  isinstance(value,  list):       # 已知各矩阵元，创建矩阵
            self.size = ( len(value),  len(value[0]))      # 矩阵的大小
            self.row =  len(value)                         # 矩阵的行数
            self.col =  len(value[0])                      # 矩阵的列数
            self.matrix = value                           # 列表储存的就是矩阵元素
```

使用嵌套列表定义矩阵后，由于 python 中列表的索引规则默认从 0 开始，当我们需要取矩阵 A 的某个矩阵元 a_{ij} 时将遇到指标不能直接对应的情况。因此我自定义了 Matrix 类的___getitem___和___setitem___方法，确保 $A[i,j]$ 可以直接提取矩阵的 a_{ij} 元。该部分代码较简单，在此不呈现出。

为了计算的方便，可定义 Matrix 类的___add___、___sub___、___mul___方法，用于计算矩阵的加减乘运算（乘法包括矩阵乘法和数乘），实现较简单，在此不陈列出代码。

对于行数 = 列数的矩阵，存在逆矩阵和行列式。为求矩阵的逆矩阵，使用初等变换法，即对原矩阵 A 和单位矩阵 I 同时进行 Gauss-Jordan 消元。为防止消元过程中误差快速传播，使用列主元法进行消元。由此可定义 Matrix 类的 inverse 方法，具体代码如下：

```
def inverse(self):      # 矩阵的逆
    assert self.row == self.col, "行数不等于列数，不能求逆"
    n = self.row
    A = copy.deepcopy(self)                     # 求逆矩阵结束，不改变原矩阵
    I = Matrix([[1 if i == j else 0 for i in  range(n)] for j in  range(n)])  # 单位矩阵
    for k in  range(1, n + 1):
        index = k
        for m in  range(k + 1, n + 1):           # 找到绝对值最大的元素作为列主元
            if  abs(A[m, k]) >  abs(A[index, k]):
                index = m       # 该元素的行指标记为index
        if A[index, k] == 0:
            return          # 逆矩阵不存在，返回None
        if index != k:
            A[k], A[index] = A[index], A[k]             # 交换两行
```

```
            I[k], I[index] = I[index], I[k]
        for i in  range(1, n + 1):              # 对第k行外的每一行进行消元
            if i != k:
                A[i, k] /= - A[k, k]
                for j in  range(k + 1, n + 1):
                    A[i, j] += A[i, k] * A[k, j]
                for j in  range(1, n + 1):
                    I[i, j] += A[i, k] * I[k, j]
        for j in  range(k + 1, n + 1):           # 主行主元归一
            A[k, j] /= A[k, k]
        for j in  range(1, n + 1):
            I[k, j] /= A[k, k]
    return I
```

使用列主元消元法可以把矩阵化为一上三角矩阵，从而可以很方便地计算矩阵的行列式。由此可定义 Matrix 类的 det 方法，具体代码如下：

```
def det(self):          # 矩阵的行列式
    assert self.row == self.col, "不是方阵，不能求行列式"
    n = self.row
    A = copy.deepcopy(self)             # 求行列式结束，不改变原矩阵
    det = 1
    for k in  range(1, n):
        index = k
        for m in  range(k + 1, n + 1):   # 找到绝对值最大的元素作为列主元
            if  abs(A[m, k]) >  abs(A[index, k]):
                index = m       # 该元素的行指标记为index
        if A[index, k] == 0:
            return 0    # 返回行列式为0
        if index != k:
            A[k], A[index] = A[index], A[k]     # 交换两行
            det *= -1                   # 行列式反号
        for r in  range(k + 1, n + 1):          # 对k行以下每一行进行消元
            A[r, k] /= - A[k, k]        # 倍乘因子
            for c in  range(k + 1, n + 1):      # 扫描该行每个元素
                A[r, c] += A[r, k] * A[k, c]            # 更新矩阵元素
        det *= A[k, k]
    det *= A[n, n]
    return det
```

为了达到结果可视化的要求，还可以定义 Matrix 类的___str___方法。在具体问题中，可以保留各矩阵元的某一给定精度，将矩阵清晰地呈现出来。在之后要解决的具体问题中，会多次将矩阵直接 print 出来，给出非常直观的结果。

此外，还可以定义 reversed 方法求矩阵的转置；norm 方法求矩阵的范数；eigenvalue 求矩阵的特征值等，不在此一一赘述。

1 桁架结构桥

1.1 解题思路分析

对于本题中的桁架结构，容易写出每个节点处的受力平衡方程：

$$\sum_{k=1}^{r} \delta_{i,i_k} N_k \cos\theta_k + \sum_{j\in I_i} F_{ij}\cos\theta_{ij} = 0 \tag{1}$$

$$\sum_{k=1}^{r} \delta_{i,i_k} N_k \sin\theta_k + \sum_{j\in I_i} F_{ij}\sin\theta_{ij} = P_i \tag{2}$$

对于这 2n 个线性方程，需要选用一种合适的方法求解之，得到桁架结构中的受力分布并加以讨论。

1. 当桥长 $l = 8$ 时，该结构中的节点数：$n = 2l = 16$；直杆数：$m = 4l - 3 = 29$；约束数：$r = 3$.

   由 $2n = 32 = m + r$ 可得：该结构满足方程存在唯一解的必要条件。

   一般的，当桥长为 l 时，也易验证方程组满足存在唯一解的必要条件。

2. 由于方程组中方程的数量 32 并不多，首先考虑用直接法求解。

   为方便标记各直杆，现对题中的桁架各节点进行编号，如图 (1) 所示：

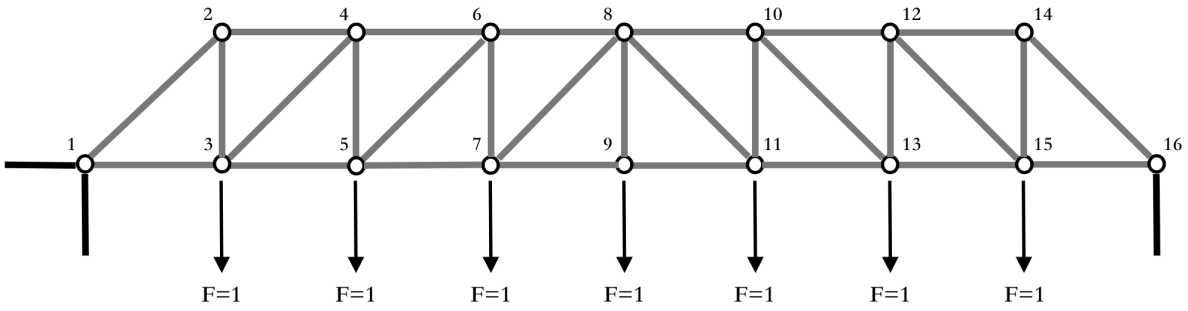


图 1: 编号示意图

对 $n = 1, 2, 3, \cdots, 16$ 的节点，分别列出受力平衡方程，如下所示：

$$N_1\cos\pi + N_2\cos\frac{3\pi}{2} + F_{12}\cos\frac{\pi}{4} + F_{13}\cos 0 = 0$$
$$N_1\sin\pi + N_2\sin\frac{3\pi}{2} + F_{12}\sin\frac{\pi}{4} + F_{13}\sin 0 = 0$$
$$F_{12}\cos\frac{5\pi}{4} + F_{23}\cos\frac{3\pi}{2} + F_{24}\cos 0 = 0$$
$$F_{12}\sin\frac{5\pi}{4} + F_{23}\sin\frac{3\pi}{2} + F_{24}\sin 0 = 0$$
$$F_{13}\cos\pi + F_{23}\cos\frac{\pi}{2} + F_{34}\cos\frac{\pi}{4} + F_{35}\cos 0 = 0$$
$$F_{13}\sin\pi + F_{23}\sin\frac{\pi}{2} + F_{34}\sin\frac{\pi}{4} + F_{35}\sin 0 = 1$$
$$F_{24}\cos\pi + F_{34}\cos\frac{5\pi}{4} + F_{45}\cos\frac{3\pi}{2} + F_{46}\cos 0 = 0$$
$$F_{24}\sin\pi + F_{34}\sin\frac{5\pi}{4} + F_{45}\sin\frac{3\pi}{2} + F_{46}\sin 0 = 0$$

$$F_{35}\cos\pi + F_{45}\cos\frac{\pi}{2} + F_{56}\cos\frac{\pi}{4} + F_{57}\cos 0 = 0$$
$$F_{35}\sin\pi + F_{45}\sin\frac{\pi}{2} + F_{56}\sin\frac{\pi}{4} + F_{57}\sin 0 = 1$$
$$F_{46}\cos\pi + F_{56}\cos\frac{5\pi}{4} + F_{67}\cos\frac{3\pi}{2} + F_{68}\cos 0 = 0$$
$$F_{46}\sin\pi + F_{56}\sin\frac{5\pi}{4} + F_{67}\sin\frac{3\pi}{2} + F_{68}\sin 0 = 0$$
$$F_{57}\cos\pi + F_{67}\cos\frac{\pi}{2} + F_{78}\cos\frac{\pi}{4} + F_{79}\cos 0 = 0$$
$$F_{57}\sin\pi + F_{67}\sin\frac{\pi}{2} + F_{78}\sin\frac{\pi}{4} + F_{79}\sin 0 = 1$$
$$F_{68}\cos\pi + F_{78}\cos\frac{5\pi}{4} + F_{89}\cos\frac{3\pi}{2} + F_{8-10}\cos 0 + F_{8-11}\cos\frac{7\pi}{4} = 0$$
$$F_{68}\sin\pi + F_{78}\sin\frac{5\pi}{4} + F_{89}\sin\frac{3\pi}{2} + F_{8-10}\sin 0 + F_{8-11}\sin\frac{7\pi}{4} = 0$$
$$F_{79}\cos\pi + F_{89}\cos\frac{\pi}{2} + F_{9-11}\cos 0 = 0$$
$$F_{79}\sin\pi + F_{89}\sin\frac{\pi}{2} + F_{9-11}\sin 0 = 1$$
$$F_{8-10}\cos\pi + F_{10-11}\cos\frac{3\pi}{2} + F_{10-12}\cos 0 + F_{10-13}\cos\frac{7\pi}{4} = 0$$
$$F_{8-10}\sin\pi + F_{10-11}\sin\frac{3\pi}{2} + F_{10-12}\sin 0 + F_{10-13}\sin\frac{7\pi}{4} = 0$$
$$F_{8-11}\cos\frac{3\pi}{4} + F_{9-11}\cos\pi + F_{10-11}\cos\frac{\pi}{2} + F_{11-13}\cos 0 = 0$$
$$F_{8-11}\sin\frac{3\pi}{4} + F_{9-11}\sin\pi + F_{10-11}\sin\frac{\pi}{2} + F_{11-13}\sin 0 = 1$$
$$F_{10-12}\cos\pi + F_{12-13}\cos\frac{3\pi}{2} + F_{12-14}\cos 0 + F_{12-15}\cos\frac{7\pi}{4} = 0$$
$$F_{10-12}\sin\pi + F_{12-13}\sin\frac{3\pi}{2} + F_{12-14}\sin 0 + F_{12-15}\sin\frac{7\pi}{4} = 0$$
$$F_{10-13}\cos\frac{3\pi}{4} + F_{11-13}\cos\pi + F_{12-13}\cos\frac{\pi}{2} + F_{13-15}\cos 0 = 0$$
$$F_{10-13}\sin\frac{3\pi}{4} + F_{11-13}\sin\pi + F_{12-13}\sin\frac{\pi}{2} + F_{13-15}\sin 0 = 1$$
$$F_{12-14}\cos\pi + F_{14-15}\cos\frac{3\pi}{2} + F_{14-16}\cos\frac{7\pi}{4} = 0$$
$$F_{12-14}\sin\pi + F_{14-15}\sin\frac{3\pi}{2} + F_{14-16}\sin\frac{7\pi}{4} = 0$$
$$F_{12-15}\cos\frac{3\pi}{4} + F_{13-15}\cos\pi + F_{14-15}\cos\frac{\pi}{2} + F_{15-16}\cos 0 = 0$$
$$F_{12-15}\sin\frac{3\pi}{4} + F_{13-15}\sin\pi + F_{14-15}\sin\frac{\pi}{2} + F_{15-16}\sin 0 = 1$$
$$N_3\cos\frac{3\pi}{2} + F_{14-16}\cos\frac{3\pi}{4} + F_{15-16}\cos\pi = 0$$
$$N_3\sin\frac{3\pi}{2} + F_{14-16}\sin\frac{3\pi}{4} + F_{15-16}\sin\pi = 0$$

为减小使用列主元消去法直接求解方程组的工作量，同时方便使用迭代法求解方程组，未知量依次取：$N_1, N_2, F_{2-4}, F_{1-2}, F_{1-3}, F_{2-3}, F_{4-6}, F_{3-4}, F_{3-5}, F_{4-5}, F_{6-8}, F_{5-6}, F_{5-7}, F_{6-7}, F_{8-10}, F_{7-8}, F_{7-9}, F_{8-9}, F_{10-12}, F_{10-11}, F_{9-11}, F_{8-11}, F_{12-14}, F_{12-13}, F_{11-13}, F_{10-13}, F_{14-16}, F_{14-15}, F_{13-15}, F_{12-15}, F_{15-16}, N_3$，使得方程组

的系数矩阵 A 对角元均非零，如图 (2) 所示。

```
[-1  0  0 0.7  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  -1  0 0.7  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  1 -0.7 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 -0.7 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   0  0  0  -1  0  0 0.7  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  1  0 0.7  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  1 -0.7 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 -0.7 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  0  0  0  -1  0  0 0.7  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  1  0 0.7  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  1 -0.7 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 -0.7 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  0  0  0  -1  0  0 0.7  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  1  0 0.7  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  1 -0.7 0  0  0  0  0 0.7  0  0  0  0  0  0  0  0
 0   0  0  0   0  0  0  0   0  0  0  0   0  0  0 -0.7 0 -1  0  0  0 -0.7 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  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  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 -1  0  0  0  1  0  0  0  0  0 0.7  0  0  0  0  0
 0   0  0  0   0  0  0  0   0  0  0  0   0  0  0 -0.7 0  0  0  1  0  0  0  0 -0.7 0  0  0  0
 0   0  0  0   0  0  0  0   0  0  0  0   0  0  0  0  -1 -0.7 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  0 0.7  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  -1  0  0  0 0.7  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  0  0  0 -0.7 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  0 0.7  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 -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 -0.7 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 0.7  0  0  0  0 -1 ]
```

图 2: l=8，方程组系数矩阵

观察图 2 中的矩阵可以发现，矩阵的每个对角元正是该列的最大元素，即列主元。同时对第 k 行以下进行高斯消元时，最多只需搜索到第 $k+5$ 行，因为 $k+6$ 行以下第 k 列的元素全为 0。结合这些特点可定义本题中求解线性方程组的 solve 函数，如下所示：

```python
def solve(A:Matrix, b:Matrix):      # 针对该问题的线性方程组解法
    for k in  range(1, A.row):
        for i in  range(k + 1,  min(k + 6, A.row + 1)):   # 对k行以下每一行进行消元，且只需搜到k+5
            A[i, k] /= - A[k, k]     # 倍乘因子
            for j in  range(k + 1, A.col + 1):       # 扫描该行每个元素
                A[i, j] += A[i, k] * A[k, j]        # 更新矩阵元素
            b[i] += A[i, k] * b[k]
    for i in  range(A.row, 0, -1):       # 回代法解方程
        for j in  range(A.col, i, -1):
            b[i] -= A[i, j] * b[j]
        b[i] /= A[i, i]
    return b        # 返回解向量
```

解出受力分布后，可使用不同颜色标记不同大小的作用力，并用 turtle 库将受力图示画出。不同大小的拉力或压力分别对应不同的 (r,g,b) 颜色通道，具体规则是：

$$(r,g,b) = \begin{cases} (\frac{1}{2}(1+F/F_{max}),0,0), & if \quad F \geq 0 \\ (0,\frac{1}{2}(1-F/F_{max}),0), & if \quad F < 0 \end{cases} \tag{3}$$

即拉力用红色表示，压力用绿色表示，颜色越纯越亮代表力的绝对值越大。代码中定义了绘制任意长度

桥梁的受力分布图示的函数 draw(l)，调用该函数可直观地看出桥梁中力的分布情况。

3. 当增加周期单元的数目时，方程组的形式仍不变。因此容易写出桥长为 l 时方程组的系数矩阵，并保证该矩阵的所有对角元均非零，从而可以使用本题定义的 solve 函数求解方程组。生成桥长为 l 时系数矩阵的代码如下：

```
def bridge_Matrix(l):       # 桥长l时待解方程组的系数矩阵
    A = Matrix((4*l, 4*l))
    A[1, 1] = -1
    A[2, 2] = -1
    for i in  range( int(l/2)):
        A[4*i + 1, 4*i + 4] = 1/sqrt(2)
        A[4*i + 2, 4*i + 4] = 1/sqrt(2)
        A[4*i + 3, 4*i + 4] = -1/sqrt(2)
        A[4*i + 4, 4*i + 4] = -1/sqrt(2)
    for i in  range( int(l/2),  int(l) - 1):
        A[4*i - 1, 4*i + 6] = 1/sqrt(2)
        A[4*i, 4*i + 6] = -1/sqrt(2)
        A[4*i + 5, 4*i + 6] = -1/sqrt(2)
        A[4*i + 6, 4*i + 6] = 1/sqrt(2)
    A[4*l - 5, 4*l - 5] = 1/sqrt(2)
    A[4*l - 4, 4*l - 5] = -1/sqrt(2)
    A[4*l - 1, 4*l - 5] = -1/sqrt(2)
    A[4*l, 4*l - 5] = 1/sqrt(2)
    for i in  range(l - 2):
        A[4*i + 3, 4*i + 3] = 1
        A[4*i + 7, 4*i + 3] = -1
    for i in  range(l - 1):
        A[4*i + 1, 4*i + 5] = 1
        A[4*i + 5, 4*i + 5] = -1
    for i in  range( int(l/2)):
        A[4*i + 4, 4*i + 6] = -1
        A[4*i + 6, 4*i + 6] = 1
    for i in  range( int(l/2),  int(l) - 1):
        A[4*i + 4, 4*i + 4] = -1
        A[4*i + 6, 4*i + 4] = 1
    A[4*l - 3, 4*l - 1] = 1
    A[4*l - 1, 4*l - 1] = -1
    A[4*l, 4*l] = -1
    return A
```

用直接法解出受力后，可讨论直杆所受最大力随桥梁长度的变化关系。

若使用迭代法求解方程组，首先需考虑迭代法能否收敛的问题。这里要用到**一阶定常迭代法基本定理**：

Thm 设有一阶定常迭代法：

$$x^{(k+1)} = Bx^{(k)} + f, \quad (k = 0, 1, 2, \cdots) \tag{4}$$

其中 $I - B$ 为非奇异矩阵，则对任意初始向量 $x^{(0)}$，迭代法得到的解序列 $\{x^{(k)}\}$ 都收敛的充要条件是谱半径 $\rho(B) < 1$，并且序列 $\{x^{(k)}\}$ 的极限 x^* 必定是方程 $x = Bx + f$ 的唯一解。

因此，我们需验证迭代矩阵 B 的谱半径小于 1。可以使用 QR 迭代算法求解 B 的特征值。该方法定义在 class Matrix 下，其主要步骤如 $Givens-Hessenberg$ 变换的代码如下所示：

```
def GHR(self):      # Givens Hessenberg Reduction
        n = self.row
        A = deepcopy(self)      # 分解结束，不改变原矩阵
        G = Matrix([[1 if i == j else 0 for i in  range(n)] for j in  range(n)])
        for j in  range(1, n):
            for i in  range(n, j, -1):
                if A[i, j]:
                    g = Matrix([[1 if i == j else 0 for i in  range(n)] for j in  range(n)])
                    g[i-1, i-1] = A[i-1, j] / sqrt(A[i-1, j]**2 + A[i, j]**2)
                    g[i, i] = A[i-1, j] / sqrt(A[i-1, j]**2 + A[i, j]**2)
                    g[i-1, i] = A[i, j] / sqrt(A[i-1, j]**2 + A[i, j]**2)
                    g[i, i-1] = -A[i, j] / sqrt(A[i-1, j]**2 + A[i, j]**2)
                    G = G * g.reverse()
                    A = g * A
        return A, G
```

当然，更方便地是使用幂法求解迭代矩阵的最大特征值 λ_m，$|\lambda_m|$ 即为 B 的谱半径。但由于求解矩阵的特征值超出这一阶段的要求，更具体的代码将在下一次大作业中给出。

以下分别对雅可比迭代、高斯-塞德尔迭代和超松弛迭代法进行讨论。雅可比迭代矩阵：$J=I-D^{-1}A$，对不同的桥长 l，计算得到的谱半径如下表所示：

l	8	10	12	14	16	18	20	22	24	26
$\rho(J)$	1.106	1.149	1.173	1.183	1.186	1.187	1.186	1.186	1.185	1.184

由此看出，$\rho(J)>1$，故雅可比迭代法不收敛，在本题中不可采用。

超松弛迭代矩阵：$M=(D-\omega L)^{-1}[(1-\omega)D+\omega U]$，高斯-塞德尔迭代矩阵 G 可令 $\omega=1$ 得到。以 $l=8$ 为例，调整不同的松弛因子 ω，计算得到的谱半径如下表所示：

ω	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2
$\rho(M)$	0.994	0.988	0.981	0.973	0.965	0.955	0.945	0.934	0.925	1.910	3.020	4.328

ω	0.80	0.81	0.82	0.83	0.84	0.85	0.86	0.87	0.88	0.89	0.9	0.91
$\rho(M)$	0.934	0.932	0.931	0.930	0.929	0.928	0.927	0.926	0.924	0.923	0.925	1.008

由以上数据可知，当松弛因子 $\omega=0.89$ 时，迭代矩阵 M 的谱半径 ρ 最小，且可以保证 $\rho<1$，这一结论对 $l\leq 20$ 总是成立的。由于一阶定常迭代的收敛速度由迭代矩阵 M 的谱半径决定，渐进收敛速度定义为：

$$R=-\log_{10}\rho(M) \tag{5}$$

故取 $\omega=1$ 可以保证迭代收敛，且收敛速度最快。但由于 M 的谱半径非常接近 1，可以预想，迭代法的收敛速度仍是很慢的。

综合上述考虑，可定义本题的线性方程组迭代解法。采用逐次超松弛法，$x0$ 为初始迭代向量；M 为最大迭代次数，本题中取 $M=1000$；e 为判断标准，当相邻解之差的列和范数小于 e 时，迭代结束；w 为松弛因子，这里将默认值设为 0.89。实现代码如下：

```
def sor(A:Matrix, b:Matrix, x0:Matrix, M: int, e: float, w=0.89):      # successive over relaxation method
    x = Matrix((A.row, 1))
    k = 0
    while k < M:    # 最多迭代M次
        x[1] = (1 - w) * x0[1] + w * (b[1] -  sum([A[1, j] * x0[j] for j in range(2, A.col+1)])) / A[1, 1]
        for i in  range(2, A.row):
            x[i] = (1 - w) * x0[i] + w * (b[i] -  sum([A[i, j] * x[j] for j in  range(1, i)]) -
                                  sum([A[i, j] * x0[j] for j in  range(i + 1, A.col + 1)])) / A[i, i]
        x[A.row] = (1 - w) * x0[A.row] + w * (b[A.row] -  sum([A[A.row, j] * x[j] for j in range(1, A.col)])) / A[A.row, A.col]
        if (x - x0).norm(0) < e:    # 已达到判停标准
            break
        x0 = deepcopy(x)       # 这样赋值必须deepcopy
        k += 1
    return x
```

4. 当小车从桥上驶过时，桁架结构中的受力情况发生连续的变化。由题意，当小车位于一根直杆上，与左右端点距离分别为 x 和 $1-x$ 时，可以等效为左右端点分别施加了 $(1-x)G$ 和 xG 的负载。为求解各杆受力分布，可以先求解只有节点 i 上存在负载 P_i 时各杆的受力分布 f_i，根据**线性叠加原理**，方程组的解即为 $f=\sum_i f_i$。这样，我们便可以求出小车处于任一位置时，桥梁中直杆所受的最大拉力。

1.2 主要实现代码

1. 定义好各个函数后，求解长为 l 的桁架结构受力分布并画出受力图示的代码很容易写出：

```
A = bridge_Matrix(l)
b = Matrix([[0 if i==2 or (i-6)%4 else 1] for i in  range(1, 4*l + 1)])
x = solve(A, b)     # 解向量中各项即代表各杆受力情况
draw(l, x)          # 画出受力示意图
```

2. 为求解直杆所受最大力随桥梁长度的变化，分别定义 maxForce_direction(l) 函数使用直接法解方程；maxForce_iteration(l, M, e, w) 函数使用迭代法解方程，二者解方程时分别使用 solve 函数和 sor 函数。使用 timeit 库中的 Timer 方法比较直接法和迭代法的计算耗时，代码如下：

```
from timeit import Timer
for l in  range(8, 22, 2):
    t1 = Timer('maxForce_direction(l)','from __main__ import maxForce_direction, l')
    t2 = Timer('maxForce_iteration(l, 1000, 0.0001*(l**2), 0.89)','from __main__ import maxForce_iteration, l')
    # 迭代法这一判停标准的选取基于对直杆最大受力的计算，见运行结果部分
    l_direction.append(t1.timeit(number=1))
    l_iteration.append(t2.timeit(number=1))
```

3. 首先将各处负载单独存在且大小为 1 时的解作为一组“基”存入列表 f 中，代码如下：

```
A = bridge_Matrix(8)
b = Matrix((32, 1))
f = []
```

```
for i in  range(1, 8):
    b[4*i + 2] = 1
    f.append(solve(deepcopy(A), deepcopy(b)))      # 将各个独立解存入列表f中
    b[4*i + 2] = 0
```

由“解题思路”中的讨论可知，当小车在任一杆直杆上，距左端点 x 距离时，各杆的受力大小一定具有 $ax+b$ 的形式。为计算直杆收到的最大拉力，我们可以将符合条件的受力以元组 (a,b) 的形式加入一集合 S 中。此处“符合条件”应理解为：$\exists x \in [0,1]$，使得 $ax+b > cx+d$，$(c,d) \in S$。当对 $\forall x \in [0,1]$，$ax+b > cx+d$，$(c,d) \in S$ 时，应在加入元素 (a,b) 的同时删去元素 (c,d)。这样对所有杆的受力都扫描结束后，集合 S 中储存的一定是使 $ax+b$ 最大的系数 (a,b)。

应该注意的是，这一部分和和上一部分有所不同，上一问待求的是以绝对值计的最大力，而这部分待求的是最大**拉力**，故只需讨论 $ax+b$ 的最大值，而非 $|ax+b|$ 的最大值。

在编写代码时，应注意第 1 根杆和最后一根杆的特殊性，这两种情况下小车重力部分作用在了约束杆上。对第 i 根杆，求最大拉力集合 S 的代码如下：

```
for i in  range(1, 9):
    print(f'当小汽车位于第{i}根杆上,距杆左端x处时，杆所受最大拉力为：',end = '')
    if i == 1:      # 小车位于第一根直杆上
        S = {(2*f[0][2]-2, 2+f[0][2]+f[1][2]+f[2][2]+f[3][2]+f[4][2]+f[5][2]+f[6][2])}    # 小车重力G对
            N_2有贡献
        j, jm = 3, 33
    elif i == 8:    # 小车位于最后一根直杆上
        S = {(-2*f[6][-1]+2, f[0][-1]+f[1][-1]+f[2][-1]+f[3][-1]+f[4][-1]+f[5][-1]+3*f[6][-1])}    # 小车
            重力G对N_3有贡献
        j, jm = 2, 32
    else:           # 小车位于中间的一根直杆
        S = {(2*f[i-1][2]-2*f[i-2][2], 2*f[i-2][2]+f[0][2]+f[1][2]+f[2][2]+f[3][2]+f[4][2]+f[5][2]+f
            [6][2])}
        j, jm = 3, 33
    while j < jm:
        if i == 1:
            t = (2*f[0][j], f[0][j]+f[1][j]+f[2][j]+f[3][j]+f[4][j]+f[5][j]+f[6][j])    # 第j根杆的受力
        elif i == 8:
            t = (-2*f[6][j], f[0][j]+f[1][j]+f[2][j]+f[3][j]+f[4][j]+f[5][j]+3*f[6][j])    # 第j根杆的受
                力
        else:
            t = (2*f[i-1][j]-2*f[i-2][j], 2*f[i-2][j]+f[0][j]+f[1][j]+f[2][j]+f[3][j]+f[4][j]+f[5][j]+f
                [6][j])
        for m in deepcopy(S):
            if m[1] >= t[1] and m[0] + m[1] >= t[0] + t[1]:
                break
            elif m[1] <= t[1] and m[0] + m[1] <= t[0] + t[1]:
                S.remove(m)
                S.add(t)
            else:
                S.add(t)
        j += 1
```

1.3 运行结果

1. $l = 8$ 时，各直杆受力情况如下：

N1	N2	F2-4	F1-2	F1-3	F2-3	F4-6	F3-4	F3-5	F4-5	F6-8	F5-6	F5-7	F6-7	F8-10	F7-8
0.0	-3.5	-3.5	-4.95	3.5	3.5	-6.0	-3.536	6.0	2.5	-7.5	-2.121	7.5	1.5	-7.5	-0.707
F7-9	F8-9	F10-12	F10-11	F9-11	F8-11	F12-14	F12-13	F11-13	F10-13	F14-16	F14-15	F13-15	F12-15	F15-16	N3
8.0	1.0	-6.0	1.5	8.0	-0.707	-3.5	2.5	7.5	-2.121	-4.95	3.5	6.0	-3.536	3.5	-3.5

draw 函数直接给出的受力图示：

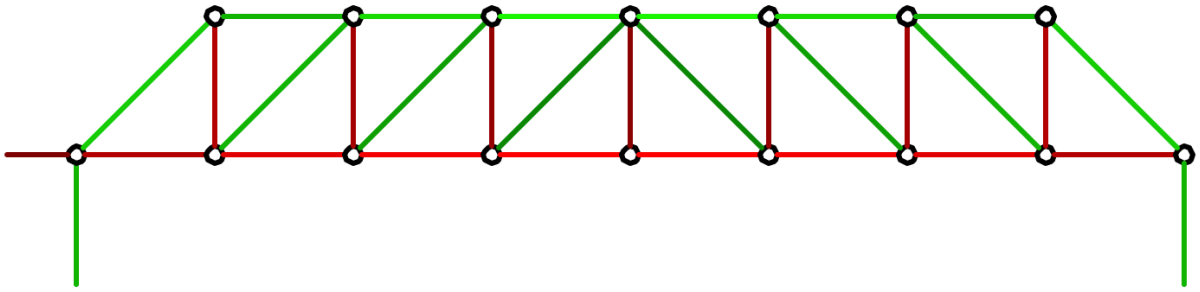


图 3: l=8 受力图示——made by draw function

更清晰的受力图示：

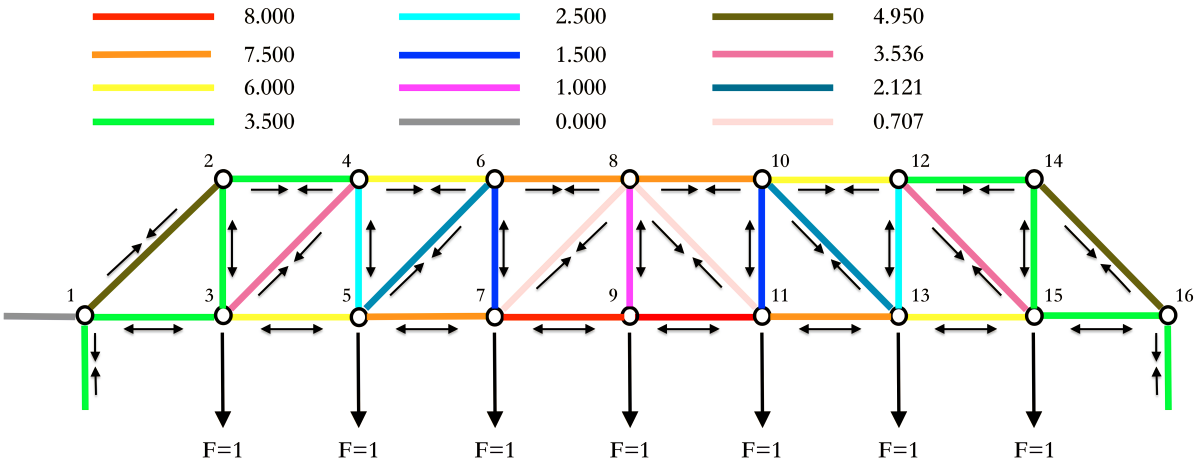


图 4: l=8 受力图示——made by hand

从上图可以看出，连接节点 7-9 和连接节点 9-11 的直杆收拉最严重，收到拉力的大小为 8；连接节点 6-8 和连接节点 8-10 的直杆收压最严重，收到压力的大小为 7.5。

2. 直杆所受最大力 (以绝对值计) 随桥梁长度的变化情况如下：

l	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38
F-max	8.0	12.5	18.0	24.5	32.0	40.5	50.0	60.5	72.0	84.5	98.0	112.5	128.0	144.5	162.0	180.5
l	40	42	44	46	48	50	52	54	56	58	60	62	64	66	68	70
F-max	200.0	220.5	242.0	264.5	288.0	312.5	338.0	364.5	392.0	420.5	450.0	480.5	512.0	544.5	578.0	612.5
l	72	74	76	78	80	82	84	86	88	90	92	94	96	98	100	102
F-max	648.0	684.5	722.0	760.5	800.0	840.5	882.0	924.5	968.0	1012.5	1058.0	1104.5	1152.0	1200.5	1250.0	1300.5

为直观地看出这个最大力和桥梁长度的变化关系，以桥梁长度 l 为横坐标，最大受力 F_{max} 为纵坐标作图，结果如图 (5) 所示：

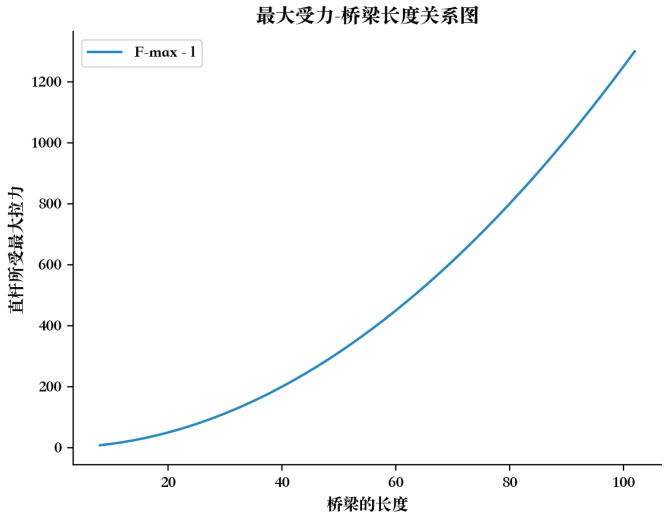


图 5: $F_{max} \sim l$

从数据和图 (5) 中都可以大致看出，$F_{max} \propto l^2$，且根据桁架结构的周期性，可以猜想下式恒成立：

$$F_{max} = \frac{1}{8} l^2, \quad \forall l = 2k \ (k \in N^+, \ k \geq 4) \tag{6}$$

直接解法和迭代解法的计算耗时比较：

桥梁长度	8	10	12	14	16	18	20
直接解法/s	0.010849	0.007701	0.011511	0.017273	0.031252	0.026709	0.032847
迭代解法/s	0.107329	0.192114	0.32794	0.597525	0.888436	1.233061	1.699571

可见，在本题中，直接解法耗时远小于迭代解法。结合之前对迭代矩阵谱半径的计算，这一点是很好理解的，因为谱半径非常接近于 1，为达到指定的精度需要非常多的迭代次数，收敛速度较慢。因此在讨论最大力随桥梁长度的变化时，选用了直接法求解方程组，这是针对本题最适合采用的方程组解法。

关于进一步增加桥梁长度时，两种解法的耗时及其计算复杂度的讨论将在“总结讨论”部分给出。

3. 桥梁中直杆所受最大拉力随小汽车位置的变化关系是：

$$F_{max} = \begin{cases} x+8, & i=1 \\ x+9, & i=2 \\ 1.25x+10, & i=3 \\ max(-0.75x+11.25,\ x+11), & i=4 \\ max(-x+12,\ 0.75x+10.5), & i=5 \\ -1.25x+11.25, & i=6 \\ -x+10, & i=7 \\ -x+9, & i=8 \end{cases} \tag{7}$$

其中每一项表示小车处于第 i 根杆上，距杆的左端点为 x 时，直杆所受最大拉力。

令 d 为小车距整个桁架左端的距离，则 F_{max} 随 d 的变化关系可写为：

$$F_{max}(d) = \begin{cases} d+8, & 0 \le d < 2 \ or \ 22/7 \le d < 4 \\ 1.25d+7.5, & 2 \le d < 3 \\ -0.75d+13.5, & 3 \le d < 22/7 \\ -d+16, & 4 \le d < 34/7 \ or \ 6 \le d \le 8 \\ 0.75d+7.5, & 34/7 \le d < 5 \\ -1.25d+17.5, & 5 \le d < 6 \end{cases} \tag{8}$$

由此可画出 $F_{max} - d$ 变化关系图，如图 (6) 所示：

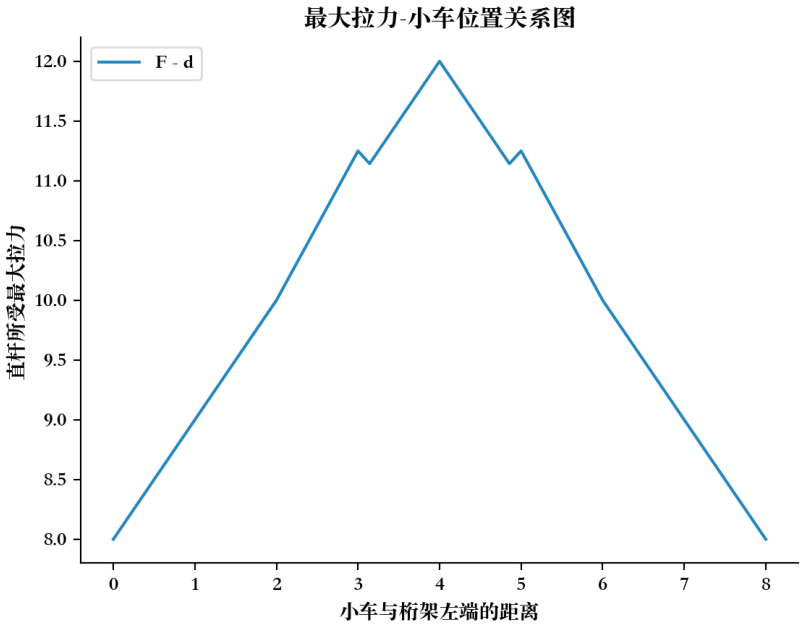


图 6: $F_{max} \sim d$

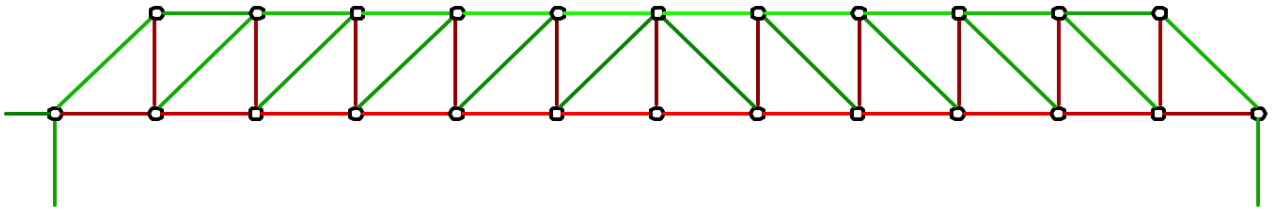
从图 (6) 中可以看出，当小车位于桥梁中点 ($d = 4$) 时，桥梁中直杆收到最大拉力，拉力值为 12。

1.4 总结讨论

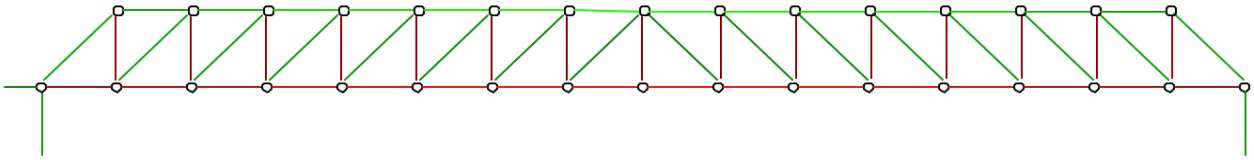
1. 关于不同长度的桥梁结构受力分布情况

   使用 draw(l,x) 函数，可以方便地画出任意长度桥梁的受力图示。下面以 $l = 12$ 和 $l = 16$ 为例，结合之前已给出的 $l = 8$ 的受力分布，讨论桁架结构受力分布特点。

   $l = 12$:



   $l = 16$:



   从图中可以发现，各直杆的受力分布和 $l = 8$ 的情形十分相似。其特点如下:

   - 受拉最严重（最红）的杆都是中间下面的两根水平杆，受压最严重（最绿）的杆都是中间上面的两根水平杆。
   - 下面的水平直杆都收拉，上面的水平直杆都收拉，受力大小从中心向两端逐渐减小。
   - 所有竖直直杆都受拉，拉力大小从中心向两端逐渐增大。
   - 所有倾斜直杆都受压，压力大小从中心向两端逐渐增大。

2. 直接解法和迭代解法计算的时间复杂度分析

   ① 使用直接解法解方程时，改进后的列主元消去法实际运算量小于一般的高斯消元法。扫描矩阵的 n 行需要对行数 k 进行一次循环，对每一个 k，需要消元的行数最多为 6，故这部分时间复杂度是 $O(1)$ 的，最后需要对列数 j 进行一次循环，进行消元。由于该过程使用了两重规模为 n 的循环，故理论分析得到：直接法解方程组的时间复杂度应为：$O(n^2)$。

   以下对 $8 \leq l < 101$ 的桥梁，分别求解线性方程组得到直杆的最大受力，每次计算重复 5 次，记下耗时与问题规模 l 的关系，代码如下:

```
1 def maxForce_direction(l):          # 直杆所受最大力的直接解法
2     A = bridge_Matrix(l)
3     b = Matrix([[0 if i==2 or (i-6)%4 else 1] for i in  range(1, 4*l + 1)])
4     x = solve(A, b)
5     force = x.norm(0)
```

```
l_direction, ll = [], [i for i in  range(8, 101, 2)]
for l in ll:
    t1 = Timer('maxForce_direction(l)','from __main__ import maxForce_direction, l')
    l_direction.append(t1.timeit(number=5))     # 重复5次计算
```

实验结果如下表所示：

l	0.038	0.047	0.065	0.109	0.133	0.132	0.161	0.202	0.225	0.269	0.317	0.364
t/s	0.427	0.484	0.532	0.594	0.674	0.735	0.802	0.877	0.951	1.056	1.140	1.272
l	1.335	1.439	1.538	1.692	1.778	1.917	2.038	2.191	2.305	2.487	2.691	2.830
t/s	2.925	3.096	3.301	3.472	3.600	3.849	4.005	4.177	4.431	4.638	4.782	

为验证该算法的时间复杂度大致为 $O(n^2)$，将该结果作图，并与二次曲线 $t = \frac{l^2}{2100}$ 比较，如图 (7) 所示：

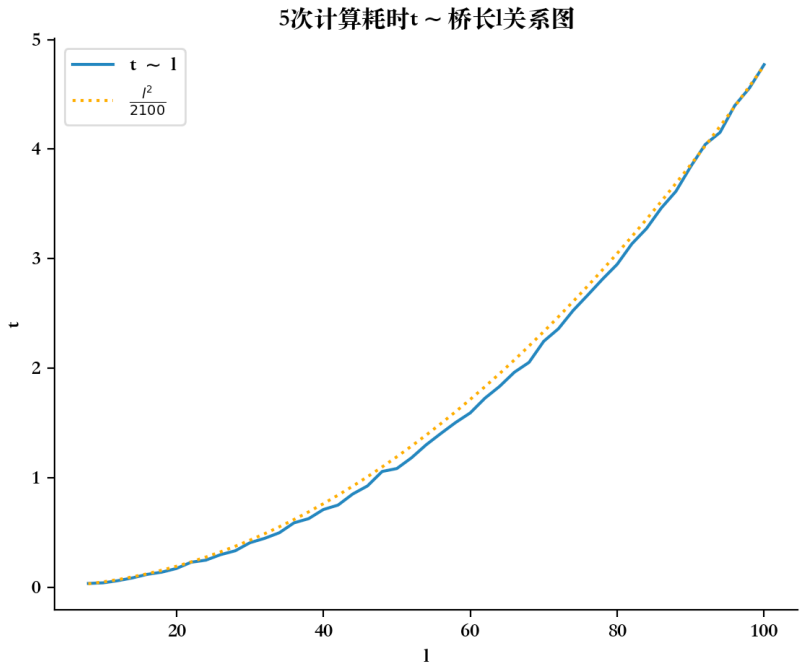


图 7: 直接解法耗时结果图

由图 (7) 可见，方程组直接解法的时间复杂度大致为 $O(n^2)$。

② 使用迭代法解方程组时，设在达到迭代次数上限 M 前，方程组解的精度已达到了指定要求（可验证，在这里计算用到的 l 的取值范围内，这一条件满足），则对每一次迭代，大约需要计算 n^2 次乘法。同时通过具体的计算可以发现，为达到指定的精度，迭代次数大致与问题的规模成正比。因此可猜想迭代解法的时间复杂度大致为 $O(n^3)$。

由于迭代解法耗时较久，以下对 $8 \le l < 41$ 的桥梁，分别求解线性方程组得到直杆的最大受力。每次计算重复 3 次，记下耗时与问题规模 l 的关系，代码如下：

```
def maxForce_iteration(l, M, e, w):     # 最大力迭代解法（桥长；最大迭代次数；判停标准；松弛因子）
    A = bridge_Matrix(l)
    b = Matrix([[0 if i==2 or (i-6)%4 else 1] for i in  range(1, 4*l + 1)])
    x = sor(A, b, Matrix((A.row, 1, 1)), M, e, w)   # 从全是1的向量开始迭代
    force = x.norm(0)
```

```
l_iteration, ll = [], [i for i in  range(8, 41, 2)]
for l in ll:
    t2 = Timer('maxForce_iteration(l, 1000, 0.0001*(l**2), 0.89)','from __main__ import
        maxForce_iteration, l')
    l_iteration.append(t2.timeit(number=3))      # 重复3次计算
```

实验结果如下表所示：

l	8	10	12	14	16	18	20	22	24
t/s	0.305	0.597	0.996	1.713	2,611	3.679	5.089	6.848	8.988
l	26	28	30	32	34	36	38	40	
t/s	11.550	14.474	18.017	22.051	26.416	31.056	36.546	42.312	

为验证该算法的时间复杂度大致为 $O(n^3)$，将该结果作图，并与三次曲线 $t = \frac{l^3}{1600}$ 比较，如图 (8) 所示：

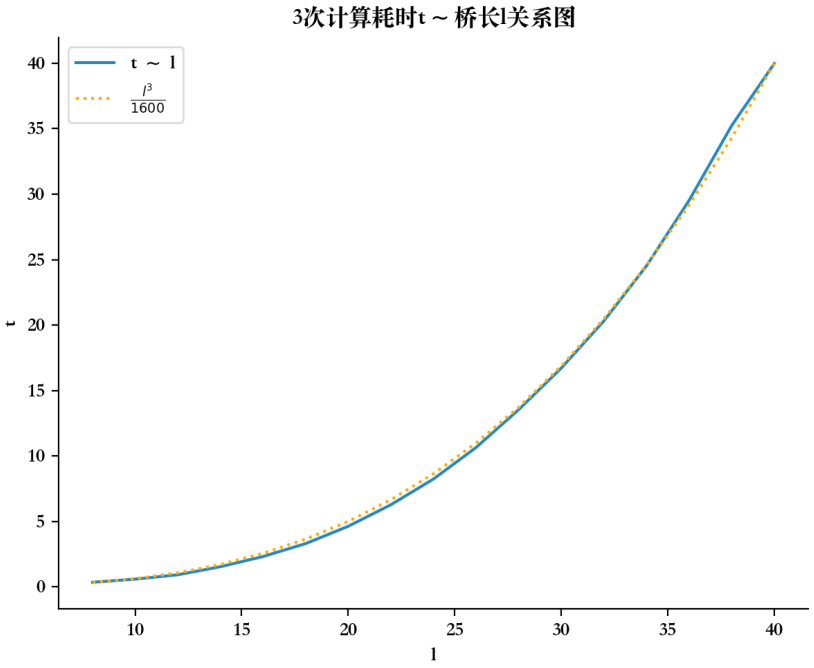


图 8: 迭代解法耗时结果图

图 (8) 中，拟合的虚线和实测的实线符合得非常好，说明当迭代次数没有达到上限 M 时，迭代解法的时间复杂度大致为 $O(n^3)$。同时通过数据结构的分析，可猜想当迭代次数达到上限后，迭代解法的时间复杂度应趋于 $O(n^2)$。当然，这是以牺牲一定的精度为代价的。

2 求解经典 dimer 模型的配分函数

2.1 解题思路分析

根据题给信息，对于行数、列数分别为 m，n 的 dimer 模型晶格，为求解其配分函数，可对其格点进行‘zigzag’编号，再构造 $mn \times mn$ 的反对称矩阵 D，将其行列式开方得到系统的配分函数 Z_{mn}。于是该问题可归结为矩阵 D 的构造和行列式的计算。

由题意，反对称矩阵 D 的非零矩阵元定义为：

$$D_{kk'} = D(r, s; r+1, s) = x \tag{9}$$

$$D_{kk'} = D(r, s; r, s+1) = y \tag{10}$$

结合‘zigzag’编号可知：

- 当 i 不是 n 的整数倍时，$D_{i,i+1} = x$，相应的 $D_{i+1,i} = -x$
- $i + j = 2kn + 1 (k \in N^+)$，当 $i < j$ 时，$D_{ij} = y$；当 $i > j$ 时，$D_{ij} = -y$
- $D_{ij} = 0, \quad others$

从而可用函数 D(m, n, x, y) 生成行数为 m，列数为 n 的反对称矩阵 D。当 x、y 为字符串时，可输出矩阵 D 的形式；当 x、y 为数时，可使用矩阵进行行列式计算。

```
def D(m, n, x, y):      # 生成矩阵D
    D = Matrix((m*n, m*n))
    for i in  range(1, m*n):
        if i % n:          # 当i为n的整数倍时，矩阵元为y或-y
            if  isinstance(x,  str):      # 输入为符号
                D[i, i+1] = 'x'
                D[i+1, i] = '-x'
            else:                         # 输入为数
                D[i, i+1] = x
                D[i+1, i] = -x
    for i in  range(1, n + 1):
        for j in  range(1, m):
            if  isinstance(y,  str):      # 输入为符号
                D[i + (j-1) * n, (j+1) * n - i + 1] = 'y'
                D[i + j * n, j * n - i + 1] = '-y'
            else:                         # 输入为数
                D[i + (j-1) * n, (j+1) * n - i + 1] = y
                D[i + j * n, j * n - i + 1] = -y
    return D
```

求解矩阵 D 的行列式可以直接调用 Matrix 类的 det 方法，其代码已在“自定义矩阵类”部分给出。

为了得到 dimer 模型的配分函数关于 x、y 的表达式，可以导入 sympy 库进行符号运算并化简得到。但根据经典 dimer 模型配分函数的定义：

$$Z_{mn}(x, y) = \sum_{\Gamma} x^{N_x} y^{N_y}, \quad N_x + N_y = \frac{mn}{2} \tag{11}$$

计算 $\sqrt{Det(D)}$ 的结果一定是关于 x、y 的 $\frac{mn}{2}$ 次齐次多项式，即：

$$Z_{mn}(x,y)=\sum_{i=0}^{\frac{mn}{2}}a_i x^i y^{\frac{mn}{2}-i}=a_{\frac{mn}{2}}x^{\frac{mn}{2}}+a_{\frac{mn}{2}-1}x^{\frac{mn}{2}-1}y+\cdots+a_1 xy^{\frac{mn}{2}-1}+a_0 y^{\frac{mn}{2}} \tag{12}$$

为确定各项系数 a_i，可通过取 $\frac{mn}{2}+1$ 组 (x,y)，计算 $\sqrt{Det(D)}$，再求解线性方程组得到。为保证算法的稳定性，减小误差传播，解线性方程组使用列主元消去法（cpe），其代码如下：

```
def cpe(A:Matrix, b:Matrix):    # 列主元消去法（column principal element)
    for k in  range(1, A.row):
        index = k
        for m in  range(k + 1, A.row + 1):   # 找到绝对值最大的元素作为列主元
            if  abs(A[m, k]) >  abs(A[index, k]):
                index = m       # 该元素的行指标记为index
        if A[index, k] == 0:
            return
        if index != k:
            A[k], A[index] = A[index], A[k]    # 交换两行
            b[k], b[index] = b[index], b[k]
        for i in  range(k + 1, A.row + 1):   # 对k行以下每一行进行消元
            A[i, k] /= - A[k, k]     # 倍乘因子
            for j in  range(k + 1, A.col + 1):   # 扫描该行每个元素
                A[i, j] += A[i, k] * A[k, j]     # 更新矩阵元素
            b[i] += A[i, k] * b[k]
    for i in  range(A.row, 0, -1):   # 回代法解方程
        for j in  range(A.col, i, -1):
            b[i] -= A[i, j] * b[j]
        b[i] /= A[i, i]
    return b      # 返回解向量
```

最后要说明的是，只有当 mn 为偶数时，经典 dimer 模型的配分函数才非零。当 mn 为奇数，即 m、n 都为奇数时，由于每个 dimer 都要占据两个格点，而一个允许的 dimer 位形中，每个格点属于且仅属于一个 dimer，因此晶格的总格点数必为偶数，这与 mn 为奇数矛盾。故对于 m 和 n 都是奇数的情况，dimer 模型配分函数必为零。

2.2 主要实现代码

1. 数值求解 dimer 模型配分函数关于 x、y 的表达式，代码如下：

```
def dimer(m, n):    # dimer模型配分函数
    A = Matrix([[(1 + 0.1*i) ** j for j in  range( int(m*n/2), -1, -1)] for i in  range(
        int(m*n/2) + 1)])
    b = Matrix([[sqrt(D(m, n, 1 + 0.1*i, 1).det())] for i in  range( int(m*n/2) + 1)])
    w = cpe(A, b)
    lw = [ int( round(w[i], 1)) for i in  range(1,  int(m*n/2) + 2)]
    Zmn = ''
    for i in  range(0,  int(m*n/2) + 1):
        if lw[i] > 0:
            if i == 0:
                Zmn += f'{lw[0]}(x^{int(m*n/2)}) '
```

```
            elif i == 1:
                Zmn += f'+ {lw[i]}(x^{int(m*n/2) - i}y) '
            elif i ==  int(m*n/2) - 1:
                Zmn += f'+ {lw[-2]}(xy^{i}) '
            elif i ==  int(m*n/2):
                Zmn += f'+ {lw[-1]}(y^{int(m*n/2)}) '
            else:
                Zmn += f'+ {lw[i]}(x^{int(m*n/2) - i}y^{i}) '
        elif lw[i] < 0:
            if i == 0:
                Zmn += f'-{-lw[0]} x^{int(m*n/2)} '
            elif i == 1:
                Zmn += f'- {-lw[i]} x^{int(m*n/2) - i}y '
            elif i ==  int(m*n/2) - 1:
                Zmn += f'- {-lw[-2]} xy^{i} '
            elif i ==  int(m*n/2):
                Zmn += f'- {-lw[-1]} y^{int(m*n/2)} '
            else:
                Zmn += f'- {-lw[i]} x^{int(m*n/2) - i}y^{i} '
    Zmn = Zmn.lstrip('+')
    return Zmn if Zmn else 0
```

2. 计算配分函数平方的解析结果，代码如下：

```
def analytical_result(m, n, x, y):      # 解析结果
    Z2 = (-1)**(m *  int(n/2)) * 2**(m * n)
    for q in  range(1, m+1):
        for r in  range(1, n+1):
            Z2 *=  complex(x * cos(pi * r / (n+1)), y * cos(pi * q / (m+1)))
    return Z2.real
```

2.3 运行结果

1. 3×4 的正方晶格对应的 D 矩阵为：

$$\begin{bmatrix}
0 & x & 0 & 0 & 0 & 0 & 0 & y & 0 & 0 & 0 & 0 \\
-x & 0 & x & 0 & 0 & 0 & y & 0 & 0 & 0 & 0 & 0 \\
0 & -x & 0 & x & 0 & y & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -x & 0 & y & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -y & 0 & x & 0 & 0 & 0 & 0 & 0 & y \\
0 & 0 & -y & 0 & -x & 0 & x & 0 & 0 & 0 & y & 0 \\
0 & -y & 0 & 0 & 0 & -x & 0 & x & 0 & y & 0 & 0 \\
-y & 0 & 0 & 0 & 0 & 0 & -x & 0 & y & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -y & 0 & x & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -y & 0 & -x & 0 & x & 0 \\
0 & 0 & 0 & 0 & 0 & -y & 0 & 0 & 0 & -x & 0 & x \\
0 & 0 & 0 & 0 & -y & 0 & 0 & 0 & 0 & 0 & -x & 0
\end{bmatrix}$$

2. 上题中经典 dimer 模型的配分函数为：

$$Z_{34}(x,y) = x^6 + 6x^4y^2 + 4x^2y^4$$

3. 进行数值结果和解析结果的比较，由于 m、n、x、y 是随机取值的，每次计算得到的具体结果不同，某一次验证得到的结果如下表所示：

i	1	2	3	4	5	6	7	8	9	10
数值结果	283.641	3.678	0.376	1.334	0.000	0.876	0.688	7.160	0.000	0.368
解析结果	283.641	3.678	0.376	1.334	0.000	0.876	0.688	7.160	0.000	0.368

运行结果截图：

```
第1次比较: m = 4, n = 5, x = 0.8735981305226025, y = 0.8104417849057641
数值计算结果为: Z^2 = 283.6409442711
解析计算结果为: Z^2 = 283.6409442711
第2次比较: m = 2, n = 6, x = 0.8392591770441014, y = 0.5991223385110366
数值计算结果为: Z^2 = 3.678454878
解析计算结果为: Z^2 = 3.678454878
第3次比较: m = 2, n = 5, x = 0.6005867555840714, y = 0.5961391205084929
数值计算结果为: Z^2 = 0.3765639893
解析计算结果为: Z^2 = 0.3765639893
第4次比较: m = 6, n = 3, x = 0.8876804799885549, y = 0.5235643353281203
数值计算结果为: Z^2 = 1.3345402657
解析计算结果为: Z^2 = 1.3345402657
第5次比较: m = 3, n = 3, x = 0.9591329697314999, y = 0.839579200377228
数值计算结果为: Z^2 = 0.0
解析计算结果为: Z^2 = 0.0
第6次比较: m = 3, n = 4, x = 0.7585620586698367, y = 0.5316634968460865
数值计算结果为: Z^2 = 0.8760534625
解析计算结果为: Z^2 = 0.8760534625
第7次比较: m = 1, n = 4, x = 0.910616858702701, y = 0.6768332068530218
数值计算结果为: Z^2 = 0.6876108888
解析计算结果为: Z^2 = 0.6876108888
第8次比较: m = 3, n = 4, x = 0.7542850458783907, y = 0.83864337230712
数值计算结果为: Z^2 = 7.160463505
解析计算结果为: Z^2 = 7.160463505
第9次比较: m = 1, n = 5, x = 0.8167743983588276, y = 0.9775708013528723
数值计算结果为: Z^2 = 0.0
解析计算结果为: Z^2 = 0.0
第10次比较: m = 4, n = 1, x = 0.7511193599316615, y = 0.7790191096312785
数值计算结果为: Z^2 = 0.3682921353
解析计算结果为: Z^2 = 0.3682921353
```

由此可见，在误差允许的范围内，数值结果和解析结果完全一致。

2.4 总结讨论

1. 任意规模经典 dimer 模型的配分函数

通过定义函数 dimer(m,n)，可以很方便地得到行数 m 和列数 n 取任意值时的配分函数关于 x, y 的表达式，如：

$$Z_{36} = x^9 + 12x^7y^2 + 20x^5y^4 + 8x^3y^6$$
$$Z_{44} = x^8 + 9x^6y^2 + 16x^4y^4 + 9x^2y^6 + y^8$$
$$Z_{45} = 9x^8y^2 + 36x^6y^4 + 37x^4y^6 + 12x^2y^8 + y^{10}$$
$$Z_{46} = x^{12} + 18x^{10}y^2 + 70x^8y^4 + 107x^6y^6 + 66x^4y^8 + 14x^2y^{10} + y^{12}$$

2. 计算的时间复杂度分析

对于给定的 x, y，考虑计算的时间复杂度随 m 或 n 的变化关系。该问题归结于计算 $mn \times mn$ 的矩阵行列式的时间复杂度。在本题中行列式使用列主元高斯消元法进行计算，时间复杂度为 $O(n^3)$，故理论分析得到计算配分函数的时间复杂度为 $O(n^3)$。

下面对该猜想进行验证。取 $m = 2$，n 依次取 $1, 2, \cdots, 20$，分别计算 $Z_{2n}(1, 1)$。重复 10 次计算，将耗时记在列表 lt 中，代码如下：

```
def calculate(m, n, x, y):          # 计算配分函数
    return sqrt(D(m,n,x,y).det())
ln, lt = [i for i in  range(1, 21)], []
m = 2
for n in ln:                        # n = 1,2,...,20
    t1 = Timer('calculate(m, n, 1, 1)','from __main__ import calculate, m, n')
    lt.append(t1.timeit(number=10))     # 10次实验耗时
```

实验结果如下表所示：

n	1	2	3	4	5	6	7	8	9	10
t/s	0.0005	0.0014	0.0046	0.0058	0.0120	0.0167	0.0225	0.0349	0.0405	0.0559
n	11	12	13	14	15	16	17	18	19	20
t/s	0.0782	0.0935	0.1138	0.1579	0.2455	0.2072	0.2982	0.2950	0.3332	0.4154

为验证该算法的时间复杂度大致为 $O(n^3)$，将该结果作图，并与三次曲线 $t = \frac{n^3}{18000}$ 比较，如图 (9) 所示：

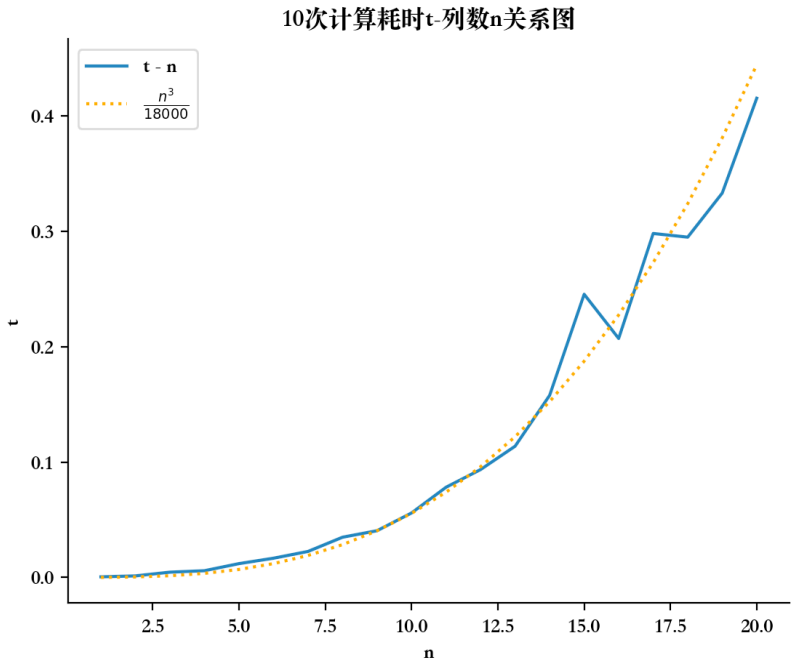


图 9: 求行列式耗时结果图

由图 (9) 可见，时间复杂度确实为 $O(n^3)$。

3 抛起的手机

3.1 解题思路分析

1. 由题意，通过数值差分给出角加速度 $\dot{\omega}_\alpha$ 后，便可得到三幅 $\dot{\omega}_\alpha \sim \omega_\beta\omega_\gamma$ 变化关系图。由欧拉运动方程：

$$\dot{\omega}_x = \frac{I_y - I_z}{I_x}\omega_y\omega_z$$
$$\dot{\omega}_y = \frac{I_z - I_x}{I_y}\omega_z\omega_x$$
$$\dot{\omega}_z = \frac{I_x - I_y}{I_z}\omega_x\omega_y$$

根据各图中数据点的变化趋势，即横轴与纵轴间是正相关或负相关可以得到各轴转动惯量的大小关系。但由于图中并不包含各时刻 t 的信息，故仅由这三幅图还无法判定手机在空中自由转动的时间段。

为得到手机在空中自由转动的时间段，即得到 $\frac{\dot{\omega}_\alpha}{\omega_\beta\omega_\gamma}$ 变化较平稳的时间段，可画出 $ln(|\frac{\beta_\alpha}{\omega_\beta\omega_\gamma}|+1) \sim t$ 关系图，取曲线较平坦的一段作为手机在空中自由转动的时间段。

2. 取定自由转动阶段后，用最小二乘法拟合可得到 $\dot{\omega}_\alpha$ 随 $\omega_\beta\omega_\gamma$ 变化的斜率 k_α。

3. 引入变量 $\tilde{x} \equiv I_x/I_z$，$\tilde{y} =\equiv I_y/I_z$，得到：

$$f_1 = \tilde{x}k_x - \tilde{y} + 1 = 0 \tag{13}$$
$$f_2 = \tilde{x} + \tilde{y}k_y - 1 = 0 \tag{14}$$
$$f_3 = -\tilde{x} + \tilde{y} + k_z = 0 \tag{15}$$

令关于 $\tilde{x}$，$\tilde{y}$ 的函数 $\varepsilon(\tilde{x},\tilde{y}) = f_1^2 + f_2^2 + f_3^2$，由于数值误差的存在，$\varepsilon$ 的值无法完全达到零，在实际问题中可以通过求 ε 的极小值来确定 $\tilde{x}$ 和 $\tilde{y}$ 的值：

$$\frac{\partial\varepsilon}{\partial\tilde{x}} = 2[(k_x^2+2)u + (-k_x+k_y-1)v - (-k_x+k_z-1)] = 0 \tag{16}$$
$$\frac{\partial\varepsilon}{\partial\tilde{y}} = 2[(-k_x+k_y-1)u + (k_y^2+2)v - (k_y-k_z+1)] = 0 \tag{17}$$

由此得到关于 $\tilde{x}$，$\tilde{y}$ 的二阶线性方程组：

$$\begin{bmatrix} k_x^2+2 & -k_x+k_y-1 \\ -k_x+k_y-1 & k_y^2+2 \end{bmatrix}\begin{bmatrix}\tilde{x}\\ \tilde{y}\end{bmatrix} = \begin{bmatrix}-k_x+k_z+1\\ k_y-k_z+1\end{bmatrix} \tag{18}$$

解之即可得到 I_x/I_z、I_y/I_z.

4. 刚体的欧拉角 (θ,φ,ψ) 与绕惯量主轴转动的角速度 $\vec{\omega}$ 的关系可写为：

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix}\cos\psi & \sin\theta\sin\psi & 0\\ -\sin\psi & \sin\theta\cos\psi & 0\\ 0 & \cos\theta & 1\end{bmatrix}\begin{bmatrix}\dot{\theta}\\ \dot{\varphi}\\ \dot{\psi}\end{bmatrix} \tag{19}$$

对每个时刻，(θ,φ,ψ) 和 $\vec{\omega}$ 都已知，通过解方程可解出该时刻的 $(\dot{\theta},\dot{\varphi},\dot{\psi})$，解方程采用列主元消去法：

```
def solve(A:Matrix, b:Matrix):    # 高斯消元法解方程
    for k in  range(1, A.row):
        index = k
        for m in  range(k + 1, A.row + 1):   # 找到绝对值最大的元素作为列主元
            if  abs(A[m, k]) >  abs(A[index, k]):
                index = m   # 该元素的行指标记为index
        if A[index, k] == 0:
            return
        if index != k:
            A[k], A[index] = A[index], A[k]    # 交换两行
            b[k], b[index] = b[index], b[k]
        for i in  range(k + 1, A.row + 1):   # 对k行以下每一行进行消元
            A[i, k] /= - A[k, k]     # 倍乘因子
            for j in  range(k + 1, A.col + 1):   # 扫描该行每个元素
                A[i, j] += A[i, k] * A[k, j]     # 更新矩阵元素
            b[i] += A[i, k] * b[k]
    for i in  range(A.row, 0, -1):   # 回代法解方程
        for j in  range(A.col, i, -1):
            b[i] -= A[i, j] * b[j]
        b[i] /= A[i, i]
    return b      # 返回解向量
```

之后用差分代替微分，更新 θ, φ, ψ 的值，得到三个欧拉角随时间的变化关系。

值得注意的是，若约定手机抛起时刻 $\theta = \varphi = \psi = 0$，则将在第一步遇到方程无解的情况。为解决该问题，可先对 θ 进行一步积分，将方程带离奇点后再数值积分，使这样可以避免积分无法继续的情况。

3.2 主要实现代码

1. 导入 matplotlib 库，作 $\dot{\omega}_x \sim \omega_y\omega_z$ 关系图代码如下（其他图类似）：

```
with  open('Gyroscope.csv', 'r') as f:
    data =  list(csv.reader(f))          # data以嵌套列表的形式存储了Gyroscope.vsc文件的所有数据
w_yz = [ eval(data[i][2]) *  eval(data[i][3]) for i in  range(1,  len(data)-1)]
beta_x = [( eval(data[i+1][1]) -  eval(data[i][1]))/ ( eval(data[i+1][0]) -  eval(data[i][0])) for i in
    range(1,  len(data)-1)]
plt.scatter(w_yz, beta_x, label = r'$\beta_x - \omega_y \omega_z$', s = 3)  # \beta_x - \omega_y\omega_z
    关系图
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')      #设置上边和右边无边框
plt.xlabel(r'$\omega_y \omega_z$')
plt.ylabel(r'$\dot{\omega}_x$')
plt.legend(loc='upper right', frameon=True)
plt.title(r'$\beta_x - \omega_y \omega_z$关系图')
plt.ion()
plt.pause(pause_time)       # 图片停留pause_time s
plt.close()
```

2. 得到手机自由转动的时间段后，最小二乘拟合的代码如下：

```
def lsr(x: list, y: list):      # Least Squares Regression
    k = ( len(x) *  sum([x[i] * y[i] for i in  range( len(x))]) -  sum(x)* sum(y)) \
        / ( len(x) *  sum([t ** 2 for t in x]) -  sum(x) ** 2)
    b = ( sum(y) - k *  sum(x)) /  len(x)
    r = ( len(x) *  sum([x[i] * y[i] for i in  range( len(x))]) -  sum(x)* sum(y)) \
        / sqrt(( len(x)* sum([t**2 for t in x]) -  sum(x)**2) * ( len(y)* sum([t**2 for t in y]) -
            sum(y)**2))
    return k, b, r      # 斜率、截距、线性相关系数
```

3. 这部分仅需求解一个二元线性方程组，比较简单。采用本题定义的 solve 函数解方程：

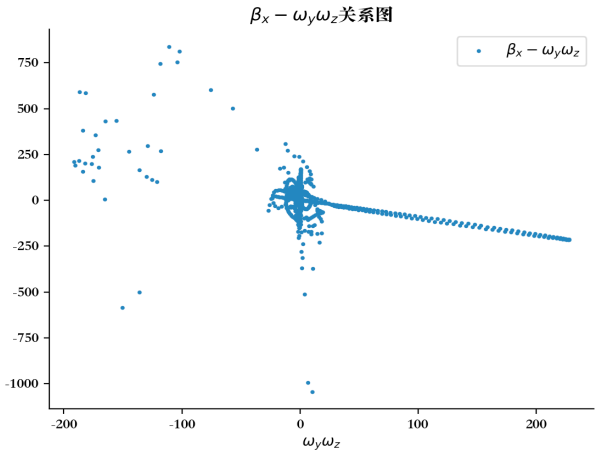
```
A = Matrix([[k_x ** 2 + 2, -k_x + k_y - 1], [-k_x + k_y - 1, k_y ** 2 + 2]])
b = Matrix([[-k_x + k_z + 1], [k_y - k_z + 1]])
I = solve(A, b)
```

4. 通过不断地解方程，更新 θ, φ, ψ，得到三个欧拉角随时间的变化关系，代码如下：

```
theta0 =  eval(data[1][1]) *  eval(data[2][0])    # 初始值，取为omega_x t
l_theta = [theta0]                  # 记录theta随时间变化关系的的列表
l_varphi = [0]                      # 记录varphi随时间变化关系的的列表
l_psi = [0]                         # 记录psi随时间变化关系的的列表
for x in lw:
    A = Matrix([[cos(l_psi[-1]), sin(l_theta[-1]) * sin(l_psi[-1]), 0],
                [-sin(l_psi[-1]), sin(l_theta[-1]) * cos(l_psi[-1]), 0],
                [0, cos(l_theta[-1]), 1]])
    Euler = solve(A, x[1])                  # 解得的是三个欧拉角对时间的导数
    l_theta.append(l_theta[-1] + Euler[1] * x[0])
    l_varphi.append(l_varphi[-1] + Euler[2] * x[0])
    l_psi.append(l_psi[-1] + Euler[3] * x[0])
l_theta = [x - theta0 for x in l_theta]     # 减去最初设定的初始值，不减问题也不大
```

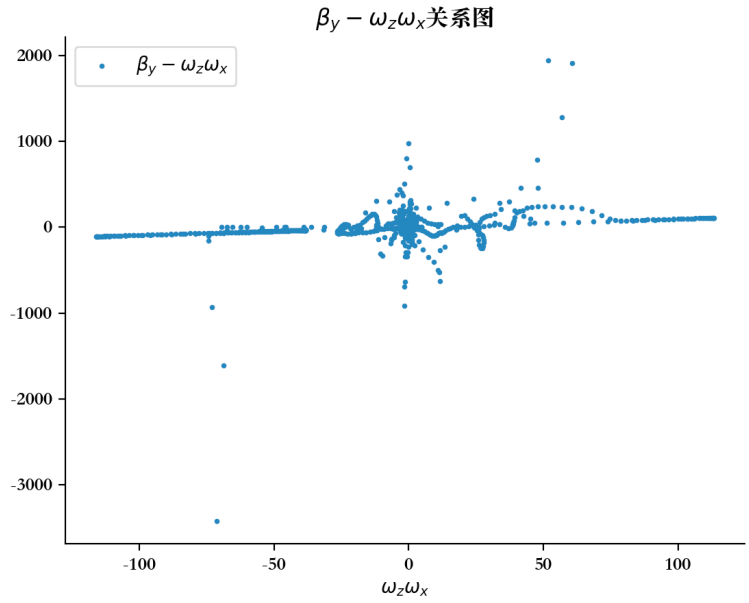
3.3 运行结果

1. $\beta_x - \omega_y\omega_z$ 关系图：



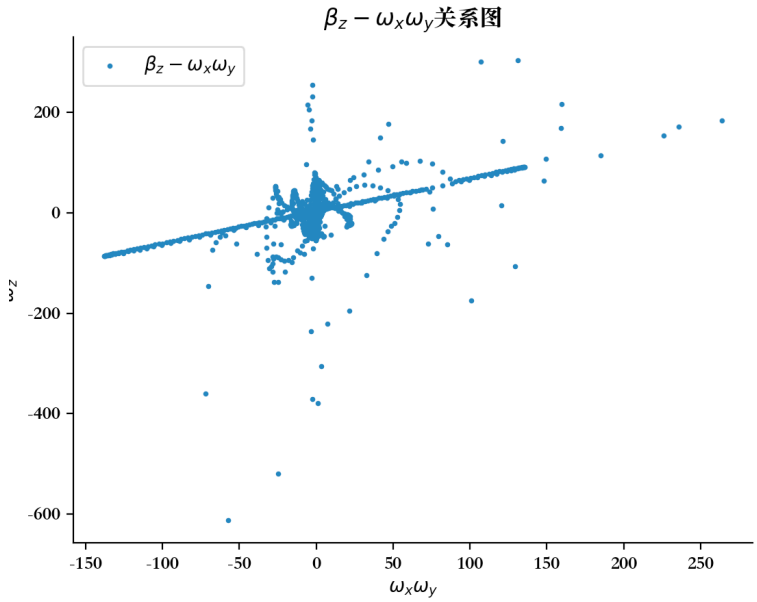
上图中，横轴与纵轴间呈负相关关系，表明 $I_y < I_z$。

$\beta_y - \omega_z\omega_x$ 关系图：



上图中，横轴与纵轴间呈正相关关系，表明 $I_z > I_x$。

$\beta_z - \omega_x\omega_y$ 关系图：



上图中，横轴与纵轴间呈正相关关系，表明 $I_x > I_y$。

从而我们得到：

$$I_z > I_x > I_y$$

为确定自由转动的时间，进一步画出 $ln(|\frac{\beta_\alpha}{\omega_\beta\omega_\gamma}|+1) \sim t$ 关系图，如图 (10) 所示。其中红色线，橙色线和蓝色线分别表示 $\alpha = x, y, z$ 的结果。

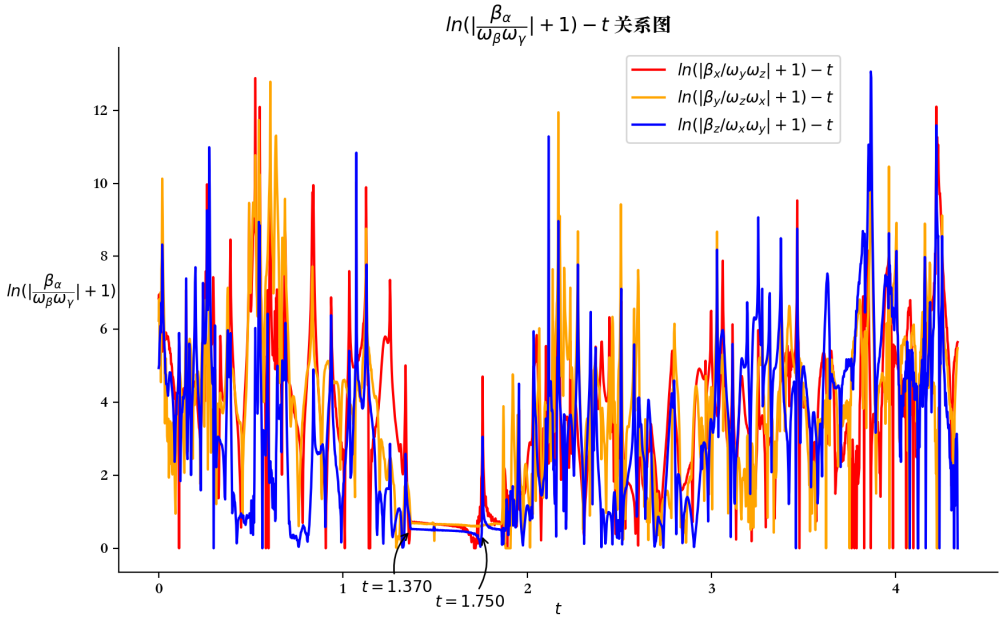


图 10: $ln(|\frac{\beta_\alpha}{\omega_\beta\omega_\gamma}|+1) \sim t$ 关系图

由图 (10) 可以清晰地看出，在 $1.37s < t < 1.75s$ 这一时间段内，对 $\alpha = x, y, z$，$\frac{\beta_\alpha}{\omega_\beta\omega_\gamma}$ 的变化都非常平缓，可近似认为保持不变。故在 $1.37s < t < 1.75s$ 这一时间段内，手机在空中自由转动。拟合时也需选取该阶段数据进行拟合，得到 k_α 的值。

2. 线性拟合的结果：$k_x = -0.957080457385997$，线性相关系数 $r_x = -0.9979194500022761$；

   $k_y = 0.9420964779512595$，线性相关系数 $r_y = 0.9997693280632908$；

   $k_z = 0.6473790900656964$，线性相关系数 $r_z = 0.9996645056163155$；

3. 可解得：

$$\tilde{x} = I_x/I_y = 0.8350857516671547$$
$$\tilde{y} = I_y/I_z = 0.18833555675482908$$

   从而手机各轴转动惯量之比为：

$$I_x : I_y : I_z = 0.8350857516671547 : 0.18833555675482908 : 1$$

4. 画出的三个欧拉角随时间变化关系图分别如图 (11)、图 (12) 和图 (13) 所示。

   由图 (11) 可以看出，θ 的变化较剧烈。尤其当手机在空中自由转动的 1.37 ∼ 1.75s 的阶段，θ 角剧烈震荡，表明这位同学没能成功让他的手机绕 z 轴稳定旋转。

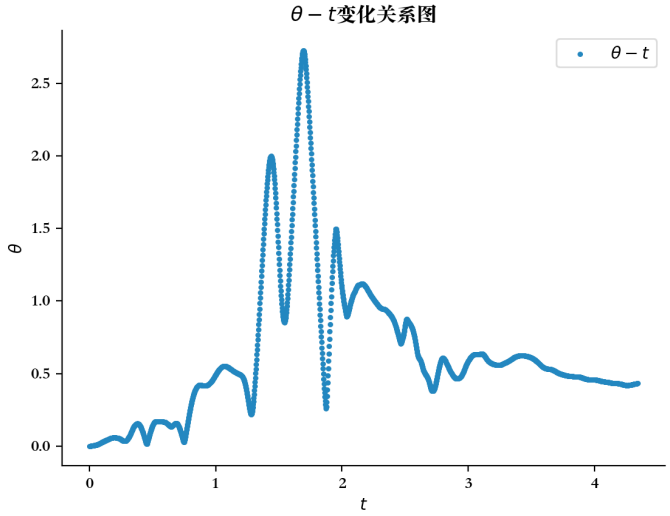


图 11: $\theta \sim t$ 变化关系图

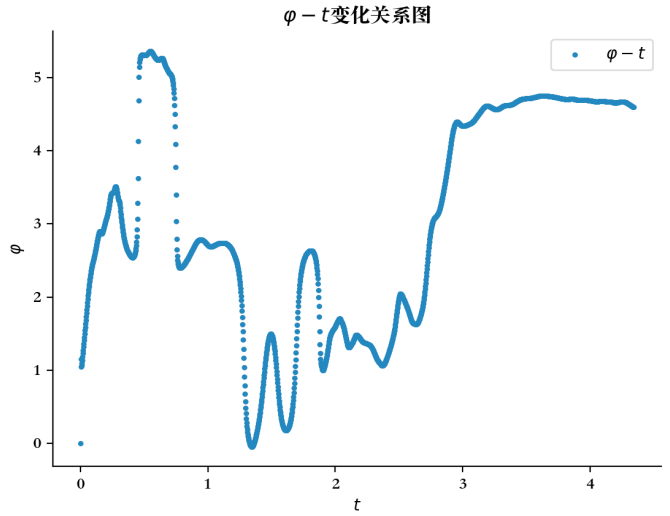


图 12: $\varphi \sim t$ 变化关系图

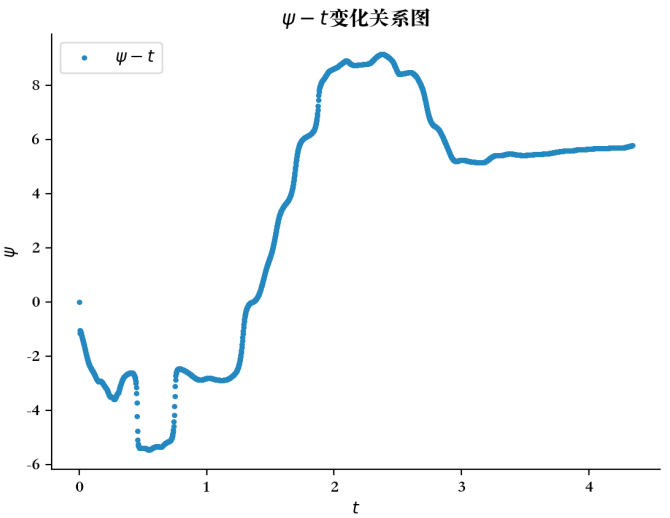


图 13: $\psi \sim t$ 变化关系图

3.4 总结讨论

1. 在计算三个欧拉角随时间的变化关系时，根据题给初始条件无法将数值积分进行下去。这里采用将方程在奇点附近作一阶展开的方法：先得到 $\theta = \omega_x t$，原则上可从第二个方程得到使 $\dot{\varphi}$ 有界的 ψ 的值，最后得到整个方程在 $t << 1$ 时的一个近似解析解，跑一个时间步长后再进行一般的数值积分。但实际运算中，我们可以先不理会 $t = 0$ 时 $\dot{\varphi}$ 和 $\dot{\psi}$ 的有界性，因为 φ 和 ψ 的初始取值可通过第一次数值积分大致确定，作图时，$t = 0$ 时刻的 $\dot{\varphi}$ 和 $\dot{\psi}$ 取值并不影响之后的结果。但初始时刻 θ 的取值对之后的积分结果有较大影响，因此必须严谨地计算 θ 在每个时刻的值，才能得到三个欧拉角正确的变化趋势。

2. 按助教意思，绕转动惯量居中的那个轴指的就是 z 轴，计算中也是将 θ 定义为关于 z 轴的夹角。但根据转动惯量的计算结果，转动惯量居中的轴实为 x 轴。若需判定手机是否绕 x 周旋转，可重新定义欧拉角的指向，重复第 4 问中的计算过程，也不难判定这位同学是否成功让他的手机绕 x 轴稳定旋转。

4 数值求解 Lippmann-Schwinger 方程

4.1 解题思路分析

1. 由于计算积分时要用到高斯求积，因此先讨论 [-1,1] 上高斯点和求积系数的生成方法。

   计算勒让德多项式 $P_l(x)$ 可利用勒让德多项式的递推关系式：

$$(2l+1)xP_l(x) = (l+1)P_{l+1}(x) + lP_{l-1}(x) \tag{20}$$

   由此可定义函数 Legendre(l, x) 计算 $P_l(x)$：

```
def Legendre(l, x):              # 勒让德多项式，l=0,1,2,···
    if l == 0:
        return 1
    elif l == 1:
        return x
    else:
        return ((2*l-1) * x * Legendre(l-1, x) - (l-1) * Legendre(l-2, x)) / l
```

   为精确计算勒让德多项式的导数 $P_l'(x)$，可利用递推关系式：

$$(x^2-1)P_l'(x) = lxP_l(x) - lP_{l-1}(x) \tag{21}$$

   由此可定义函数 Legendre_diff(l, x) 计算 $P_l'(x)$：

```
def Legendre_diff(l, x):         # 勒让德多项式的导数，l=0,1,2,···
    if l == 0:
        return 0
    else:
        return (l * Legendre(l-1, x) - l * x * Legendre(l, x)) / (1 - x**2)
```

   为计算 $P_l(x)$ 在 $[-1,1]$ 上的所有零点，可先用牛顿法找出方程的某个根。使用牛顿法的理由是：我们事先并不知道函数在哪两个点函数值反号，而牛顿法可以从任一点出发，搜到方程的某个根。因此可以不断使用牛顿法从 $x0=-1$ 处开始求根。根据牛顿法的迭代公式：

$$x^{(k+1)} = x^{(k)} - \frac{1}{f'(x^{(k)})} f(x^{(k)}) \tag{22}$$

   容易写出函数 Newton_Method(f, x0)：

```
def Newton_Method(f, x0):        # 牛顿法找x0附近的零点
    diff = lambda f,x: (f(x+(10**-5)) - f(x-(10**-5))) / (2*(10**-5))   # 差商型求导
    x = x0
    while  abs(f(x0)) > (10 ** -6) or  abs(x0 - x) > (10 ** -6):
        x = x0 - f(x0) / diff(f, x0)     # x存储的是x_{k+1}
        x0, x = x, x0       # 交换后，x0代表迭代得到的最新结果，x代表上一部迭代结果
    return x0
```

   求出 $P_l(x)$ 的某一个零点 x_1 后，可将原 l 次多项式收缩为低一次的 $l-1$ 次多项式 $P_l(x)/(x-x_1)$，如此不断地求出 $P_l(x)$ 的所有 l 个零点，即为 l 个高斯点。以下代码将 n 阶勒让德多项式产生的 n 个高斯点存入列表 lp 中：

```
f = lambda x: Legendre(n, x)          # 给定n，生成n阶勒让德多项式
lp = [Newton_Method(f, -1)]           # lp存储n个高斯点
for _ in  range(n-1):
    def g(x):
        if x in lp:
            g_value = Legendre_diff(n, x)       # 洛必达法则
            for t in lp:
                g_value /= (x - t) if t != x else 1     # 除掉已得到的因子
        else:
            g_value = Legendre(n, x)
            for t in lp:
                g_value /= (x - t)      # 除掉已得到的因子
        return g_value
    lp.append(Newton_Method(g, -1))
```

得到高斯点列表 lp 后，需还计算 n 个求积系数。先写出拉格朗日插值基函数：

```
def basic_func(lx: list, k, x):         # 拉格朗日插值基函数
    l = 1
    for t in lx:
        l *= 1 if t == lx[k] else (x - t) / (lx[k] - t)
    return l
```

积分使用精度较高的复化 cotes 积分公式：

```
def complex_cotes(f, a, b, n):          # 复化柯特斯积分公式
    h = (b - a) / n
    x1 = [f(a + (i+1/4)*h) for i in  range(n)]
    x2 = [f(a + (i+1/2)*h) for i in  range(n)]
    x3 = [f(a + (i+3/4)*h) for i in  range(n)]
    x4 = [f(a + i*h) for i in  range(1, n)]
    C = h / 90 * (7*f(a) + 32* sum(x1) + 12* sum(x2) + 32* sum(x3) + 14* sum(x4) + 7*f(b))
    return C
```

对第 k 个高斯点，其求积系数 A_k 由下式计算：

$$A_k = \int_{-1}^{1} l_k(x)dx \tag{23}$$

以下代码将 n 个高斯点对应的求积系数存入列表 lc 中：

```
lc = []                             # lc存储n个求积系数
    for i in  range(n):
        h = lambda x: basic_func(lp, i, x)
        C = complex_cotes(h, -1, 1, 64)      # 使用64阶cotes公式计算积分值
        lc.append(C)
```

至此，具有 $2n+1$ 阶代数精度的高斯积分公式对应的高斯点和求积系数已全部获得。

2. 得到 $[-1,1]$ 上的高斯点后，还需对题中的积分区间进行变换，积分式：

$$\begin{aligned}\langle E'|T|E_0\rangle = \langle E'|V|E_0\rangle + (\int_0^{E_0-\varepsilon} + \int_{E_0+\varepsilon}^{\infty})dE'' \frac{\langle E'|V|E''\rangle\langle E''|T|E_0\rangle}{E_0 - E''} \\ - i\pi\langle E'|V|E_0\rangle\langle E_0|T|E_0\rangle\end{aligned} \tag{24}$$

利用高斯求积将积分化为求和：

$$\begin{aligned}\langle E'|T|E_0\rangle = \langle E'|V|E_0\rangle + \sum_{j=1}^{n}\omega_j \frac{\langle E'|V|E_j\rangle\langle E_j|T|E_0\rangle}{E_0 - E_j} + \sum_{j=n+1}^{2n}\omega_j \frac{\langle E'|V|E_j\rangle\langle E_j|T|E_0\rangle}{E_0 - E_j} \\ - i\pi\langle E'|V|E_0\rangle\langle E_0|T|E_0\rangle\end{aligned} \tag{25}$$

当 $j = 1, 2, \cdots, n$ 时，ω_j 和 E_j 需由积分区间 $[0, E_0]$ 变换到 $[-1, 1]$ 上得到。令：

$$f(E'') = \frac{\langle E'|V|E''\rangle\langle E''|T|E_0\rangle}{E_0 - E''} \tag{26}$$

作代换：

$$E''(x) = \frac{E_0}{2}(1+x) \implies \int_0^{E_0} f(E'')dE'' = \frac{E_0}{2}\int_{-1}^{1} f[E''(x)]dx \tag{27}$$

设已求得的高斯点为 x_j，相应的求积系数为 A_j，则上述积分可化为求和：

$$\frac{E_0}{2}\int_{-1}^{1} f[E''(x)]dx = \sum_{j=1}^{n}\frac{E_0}{2}A_j f(\frac{E_0}{2}(1+x_j)) = \sum_{j=1}^{n}\omega_j f(E_j) \tag{28}$$

$$\implies \omega_j = \frac{E_0}{2}A_j \ ; \ E_j = \frac{E_0}{2}(1+x_j) \ , \ j = 1, 2, \cdots, n \tag{29}$$

当 $j = n+1, n+2, \cdots, 2n$ 时，ω_j 和 E_j 需由积分区间 $[E_0, \infty]$ 变换到 $[-1, 1]$ 上得到。选取合适的 u 值，作代换：

$$E''(x) = \frac{2E_0 + u(1+x)}{1-x} \implies \int_{E_0}^{\infty} f(E'')dE'' = \int_{-1}^{1}\frac{2(E_0+u)}{(1-x)^2}f[E''(x)]dx \tag{30}$$

设已求得的高斯点为 x_j，相应的求积系数为 A_j，则上述积分可化为求和：

$$\int_{-1}^{1}\frac{2(E_0+u)}{(1-x)^2}f[E''(x)]dx = \sum_{j=1}^{n}\frac{2(E_0+u)}{(1-x_j)^2}A_j f(\frac{2E_0+u(1+x_j)}{1-x_j}) = \sum_{j=n+1}^{2n}\omega_j f(E_j) \tag{31}$$

$$\implies \omega_j = \frac{2(E_0+u)}{(1-x_{j-n})^2}A_{j-n} \ ; \ E_j = \frac{2E_0+u(1+x_{j-n})}{1-x_{j-n}} \ , \ j = n+1, n+2, \cdots, 2n \tag{32}$$

这样便可以将所有 ω_j 和 E_j 用高斯点表示出来。若有 n 个高斯点，则 j 可取 $N = 2n$ 个值。

3. 定义了矢量 $[T]$，矩阵 F 和矢量 $[V]$ 后，可使用列主元消去法解方程组：$F[T] = [V]$，得到的 $[T]$ 的最后一个分量 $T_{N_1N_1}$ 正比于 S 波的散射振幅，由此可讨论微分散射截面随初始能量 k_0 的变化关系。

4. 由于'delta shell' 势的 S 波散射具有如下解析结果：

$$[]e^{2i\delta_0} = \cos 2\delta_0 + i \sin 2\delta_0 = \frac{1 + \frac{2m\gamma}{\hbar^2 k} e^{-ika} \sin ka}{1 + \frac{2m\gamma}{\hbar^2 k} e^{ika} \sin ka} = Z(k) \tag{33}$$

从而：

$$|\sin \delta_0|^2 = \frac{1 - \cos 2\delta_0}{2} = \frac{1 - Re[Z(k)]}{2} \tag{34}$$

改变 k 的值，便可研究 $|\sin \delta_0|^2$ 随 k 的变化关系。增大 γ 的值，可验证共振点的位置大致满足：$ka = n\pi$。

4.2 主要实现代码

1. 求解矩阵方程的代码：

```
def solve(k0, gamma, a, lp, lc):      # 求解矩阵方程
    mid = 10
    E0 = k0 ** 2
    n =  len(lp)           # 高斯点的个数
    N = 2*n               # 对N个点做高斯求积
    lw = [E0 / 2 * lc[i] for i in  range(n)] + [2 * (E0 + mid) / (1-lp[i])**2 * lc[i] for i in
        range(n)]    # 存储w_j, 其中j=1,2,...,N
    lE = [E0 / 2 * (1 + lp[i]) for i in  range(n)] + [(2*E0 + mid*(1+lp[i])) / (1-lp[i]) for i in
        range(n)]
    lE.append(E0)       # 存储E_j, 其中j=1,2,...,N
    lD = [lw[i] / (E0 - lE[i]) for i in  range(N)]                    # 存储D_j, 其中j=1,2,...,N
    lD.append( complex(0, -pi))
    func_delta = lambda i, j: 1 if i == j else 0
    func_V = lambda i, j: 4*gamma * sin(sqrt(lE[i-1])*a) * sin(sqrt(lE[j-1])*a) / (lE[i-1]*lE[j-1])
        **(1/4)
    F = Matrix((N+1, N+1))                                           # 系数矩阵F
    V = Matrix([[func_V(i, N+1)] for i in  range(1, N+2)])           # 右端向量[V]
    for i in  range(1, N+2):
        for j in  range(1, N+2):
            F[i, j] = func_delta(i, j) - lD[j - 1] * func_V(i, j)
    T = cpe(F, V)           # 列主元消去法解方程组
    return T                # 返回待求向量[T]
```

绘制 $|\langle E_0|T|E_0\rangle|^2$ 随 k_0 变化的曲线代码：

```
lk0, lT = [i/50 for i in  range(1, 501)], []
for k0 in lk0:
    T0 = solve(k0, gamma, a, lp, lc)[2*n+1]
    lT.append(( abs(T0)) ** 2)
plt.plot(lk0, lT, label = r'$|\langle E_0|T|E_0 \rangle|^2 - k_0$')
plt.xlabel(r'$k_0$')
plt.ylabel(r'$|\langle E_0|T|E_0 \rangle|^2$')
plt.legend(loc='upper right', frameon=True)
plt.title(rf'$|\langle E_0|T|E_0 \rangle|^2 - k_0$变化曲线 ($a = 1, \gamma = {gamma}$)')
plt.show()
```

4.3 运行结果

1. 取 $\gamma = 5$，对 $k_0 \in (0, 10)$，求解矩阵方程，得到的 $|\langle E_0|T|E_0\rangle|^2$ 随 k_0 变化的曲线如图 (14) 所示。

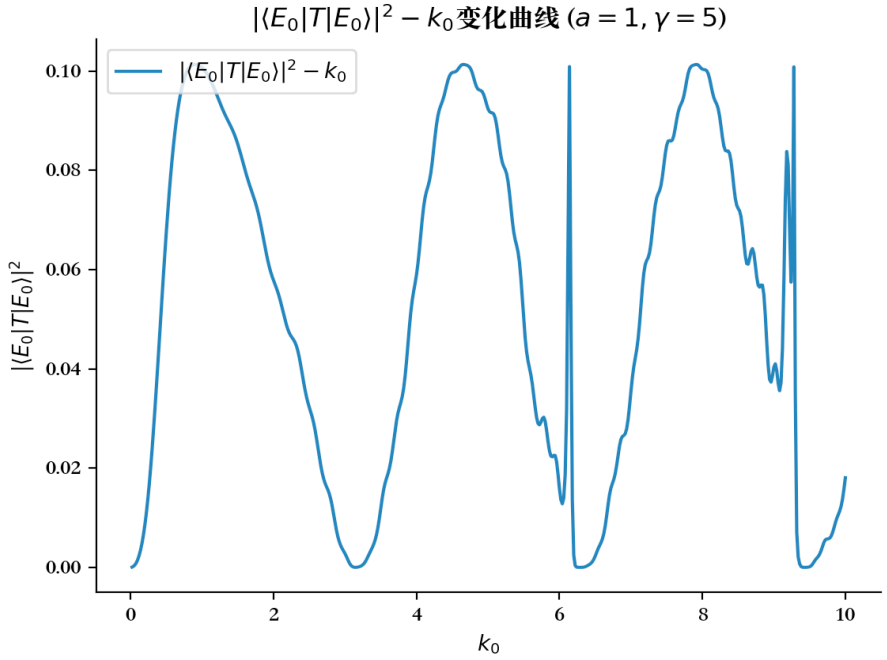


图 14: $|\langle E_0|T|E_0\rangle|^2 \sim k_0$ 关系图

图中可观察到两个共振点，读出它们对应的 k_0 分别为：6.14、9.27。

2. 取 $\gamma = 25$，对 $k \in (0, 10)$，用解析结果画出的 $|\sin\delta_0|^2$ 随 k 变化关系的曲线如图 (15) 所示。

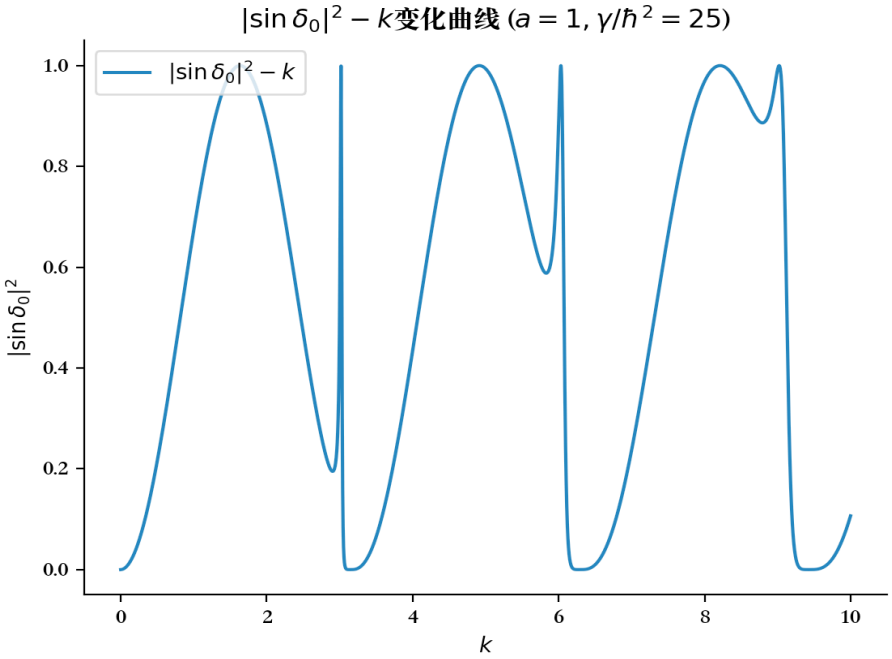


图 15: $|\sin\delta_0|^2 \sim k$ 关系图

图中可观察到三个共振点，读出它们对应的 k 分别为：3.02、5.98、9.03。

3. 当 γ 很大时，不妨取 $\gamma = 200$，$|\sin\delta_0|^2$ 随 k 变化关系的曲线如图 (16) 所示。

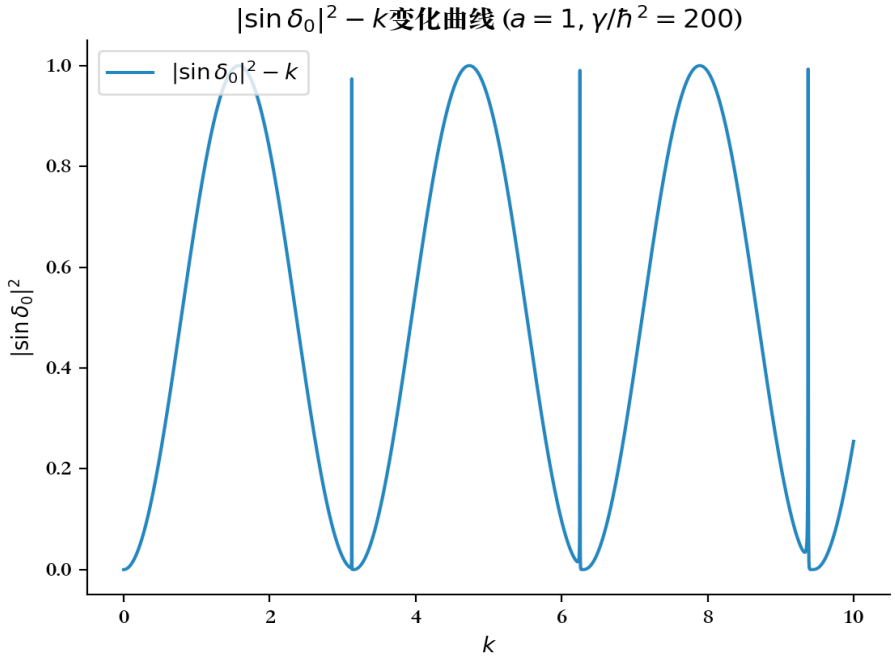


图 16: $|\sin\delta_0|^2 \sim k$ 关系图

由图 (16) 可见，当 $\gamma = 200$ 很大时，共振点约为 $k_1 = 3.13 \approx 3.14 = \pi$、$k_2 = 6.25 \approx 6.28 = 2\pi$、$k_3 = 9.39 \approx 9.42 = 3\pi$，即共振点的 k 值大致满足：$ka = n\pi$。

4.4 总结讨论

1. 使用高斯求积将积分化为求和时，若没有把积分区间分为两段，而是直接将 $(0, +\infty)$ 上的积分变换到 $(-1, 1)$ 上，则使用数值求解矩阵方程的方法会出现散射截面随 k_0 快速震荡的情况（如图 (17) 所示）。这是由于对某些 k 值，用高斯点取出的某项 E_{j0} 和 E_0 非常接近，导致这一项对积分值的贡献异常大。

2. 解析结果给出，S 波散射的相移满足 (33) 式，当 $Z(k)$ 的分母为零时，给出共振点的位置满足：

$$e^{2ika} = 1 - 2i\frac{\hbar^2 k}{2m\gamma} \tag{35}$$

只有当 k 满足：$ka = n\pi$ 时，上式才能严格成立，即当 $\gamma \to \infty$ 时，共振点位置精确为 $ka = n\pi$。

当 γ 为有限值时，只有当 ka 在 $n\pi$ 附近，且 γ 较大时，分母才足够小，在散射振幅随能量的变化曲线中才能观察到共振点的存在。这样分析得到的结果和程序运行得到的结果基本一致。

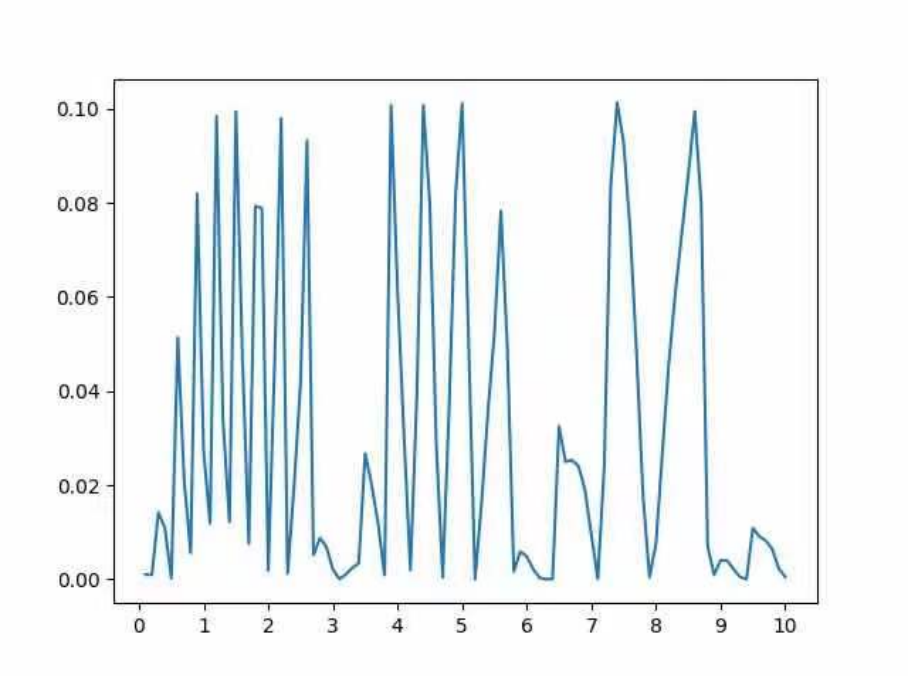


图 17: 高斯点选取不当使结果剧烈震荡图

附录

A python 程序运行环境

- 操作系统：Mac OS X Catalina 10.15
- 处理器：1.6 GHz 双核 Intel Core i5
- 内存：8 GB 2133 MHz LPDDR3
- 图形卡：Intel UHD Graphics 617 1536 MB
- python 解释器：Python 3.9.4

B 大作业中用到的课堂知识

- 三角形方程组的解法
- 矩阵的 LU 分解和列主元 LU 分解
- 列主元高斯消元法与矩阵行列式的计算
- 高斯-约当消元法与逆矩阵的计算
- 向量和矩阵的范数
- 雅可比迭代法和高斯-赛德尔迭代法
- 经典迭代法的收敛判定
- 拉格朗日插值法
- 差商形式的数值微分
- 插值形式的数值积分
- 变步长积分——复化积分法
- 高斯积分
- 非线形方程求根——牛顿法
- 矩阵特征值的计算（相关代码将在下一次大作业进一步完善）