

计算物理第二次大作业

王台宜

学号：1500011419

2019年5月5日

目录

1 第一题	3
1.1 题目	3
1.2 分析与思路	4
1.2.1 第一问 (1) & 第二问 (2)	4
1.2.2 第三问 (3) & 第四问 (4)	4
1.3 算法描述和实现	5
1.3.1 第 (1) (2) (3) 问	5
1.3.2 第 (4) 问	11
1.4 结果展示	14
1.4.1 (1)	14
1.4.2 (2)	15
1.4.3 (3)	15
1.4.4 (4)	17
2 第二题	19
2.1 题目	19
2.2 分析与思路	21
2.2.1 第一问 (1)	21
2.2.2 第二问 (2) & 第三问 (3)	22
2.3 算法描述和实现	23
2.3.1 预先定义的函数	23
2.3.2 第 (1) 问	25
2.3.3 第 (2) 问 & 第 (3) 问	26
2.4 结果展示	28
2.4.1 (1)	28
2.4.2 (2)	29
3 第三题	30
3.1 题目	30
3.2 问题 (1)	30

3.2.1	解	30
3.3	问题 (2)	30
3.3.1	思路	30
3.3.2	程序实现	31
3.3.3	结果	31
3.4	问题 (3)	33
3.4.1	思路	33
3.4.2	程序实现	33
3.4.3	结果展示	35
3.5	问题 (4)	40
3.5.1	解	41
3.6	问题 (5)	41
3.6.1	解	41
3.7	问题 (6)	41
3.7.1	解	41
3.8	问题 (7)	42
3.8.1	解	42
3.9	问题 (8)	42
3.9.1	解	42

Chapter 1

第一题

1.1 题目

(一) 【45分】马丢方程(Mathieu equation)是一类重要的微分方程，其解为马丢函数(Mathieu function)。马丢函数在很多物理场景中都有着重要的应用，例如：椭圆鼓的振动，射频四极矩，浮体的稳定性，带电粒子的保罗阱(Paul trap)，等等。这方面的科普介绍，有兴趣的同学可参考：Lawrence Ruby, *Am. J. Phys.* **64**, 39 (1996), “*Applications of the Mathieu equation*”; 以及Wolfgang Paul在1989年获得诺贝尔奖时的演讲，发表在*Rev. Mod. Phys.* **62**, 531 (1990), “*Electromagnetic traps for charged and neutral particles*”。正则形式的马丢方程为：

$$-\frac{d^2\Psi(\phi)}{d\phi^2} + 2q \cos(2\phi)\Psi(\phi) = A\Psi(\phi), \quad (1)$$

其中 q 为参数， A 为本征值。准确快速地求解上述方程的本征值和本征矢在马丢函数的物理应用中非常重要。本题中，我们来研究一种基于离散变量表示法(discrete variable representation)的方法来求解。鉴于马丢函数的周期性，我们考虑如下傅里叶基矢

$$\Phi_n(\phi) = \frac{\exp^{in\phi}}{\sqrt{2\pi}}, \quad (2)$$

其中 $n = -M, -(M-1), \dots, 0, \dots, M-1, M$ ，且 M 为正整数。我们取离散格点

$$\phi_k = \frac{2\pi}{2M+1}k, \quad (k = 1, 2, \dots, 2M+1), \quad (3)$$

构造基函数

$$f_k(\phi) = \frac{1}{2M+1} \sum_{n=-M}^{n=M} \Phi_n^*(\phi_k) \Phi_n(\phi), \quad (4)$$

则我们可以将(1)中的本征函数 $\Psi(\phi)$ 用 $f_k(\phi)$ 来展开，亦即

$$\Psi(\phi) = \sum_{k=1}^{2M+1} C_k f_k(\phi), \quad (5)$$

可以证明，在 $f_k(\phi)$ 基函数的展开下，(1)中的“动能项” $(-\frac{d^2}{d\phi^2})$ 的矩阵元 T_{jk} 可以由下式给出

$$T_{jk} = \begin{cases} (-1)^{j-k} \frac{\cos[\pi(j-k)/(2M+1)]}{2\sin^2[\pi(j-k)/(2M+1)]}, & j \neq k; \\ \frac{M(M+1)}{3}, & j = k. \end{cases} \quad (6)$$

同时，可以证明，此展开下，(1)中的“势能项” $2q \cos(2\phi)$ 的矩阵元 V_{jk} 近似为对角的，即

$$V_{jk} \approx 2q \cos(2\phi_j) \delta_{jk}. \quad (7)$$

因此，求解(1)的本征值问题，转化如下本征值问题：

$$\mathbf{H}\Psi = [\mathbf{T} + \mathbf{V}]\Psi = \mathbf{A}\Psi. \quad (8)$$

显然， \mathbf{H} 为实对称稠密矩阵，非常适合利用QR算法来求解本征值问题。

1. 请写出最一般的**实对称矩阵的实用QR算法**(讲义第7讲的倒数第2、3页)。注意算法中包括了Householder-Hessenberg约化、Givens旋转变换、以及原点位移等。
2. 取 $M = 50$ ，即矩阵维数 $N = 2M + 1 = 101$ ，对参数 $q \in [0, 20]$ 计算头11个本征值，不用展现数值结果，只用作图画在一张 (q, A) 图上。注意， q 区间的端点必须包括，间距 $\Delta q = 1.0$ 。
3. 马丢方程的本征函数有奇宇称和偶宇称解。为了考察你的解对格点数的收敛性，请分别取 $M = 5$ 和 $M = 40$ ，对比两种格点下算出的头5个偶宇称解的本征值。也只用画在一张 (q, A) 图上。参数 $q \in [0, 20]$ ， q 区间的端点必须包括，间距 $\Delta q = 1.0$ 。
4. 最后，对 $M = 50$ 、 $q = 10.0$ 的情况，请求出头6个偶宇称的本征值(列表给出)和本征矢 $\Psi_i^{(even)}(\phi)$ (在 $\phi \in [0^\circ, 90^\circ]$ 间一起画图给出)。

1.2 分析与思路

1.2.1 第一问 (1) & 第二问 (2)

- 利用式 (6) 对矩阵进行初始化。
- 按照讲义里的 QR 算法进行计算：先进行 Householder 变换，把矩阵变为上三角矩阵；接着进行 Givens 旋转，消去对角线下元素；过程加入原点位移加速收敛。由于 A 是对称矩阵，经过 Householder 变换后，矩阵转为对称三对角矩阵。
- 实际计算中，其他元素并不是严格为 0，所以加个判定条件，即若矩阵元素小于 10^{-10} 就设这个数为 0。
- 通过以上变换， A 最终被变换得到上三角矩阵，其对角元素即为原矩阵本征值。所得上三角矩阵即为所求的分解得到的 R 矩阵。其中在 Householder 变换过程中， A 首先经过了 $n-2$ 个初等反射阵 $U_{n-2} \dots U_1$ 的累乘，又在 Givens 旋转变换过程中，经过了 $n-1$ 个初等旋转矩阵 $G_{n-1} \dots G_1$ 的累乘，最终得到 $A = QR$ 的分解，而 Q 矩阵必然就是 $Q = U_1 U_2 \dots U_{n-2} G_1^{-1} G_2^{-1} \dots G_{n-1}^{-1}$ 。

1.2.2 第三问 (3) & 第四问 (4)

在第一问和第二问的基础上，第三问需要寻找方法来求出“偶宇称”的解。此处想从方程的本征函数入手，将题中 (4) 和 (5) 式合并，可以得到方程：

$$\begin{aligned} \Phi(\phi) &= \sum_{k=1}^{2M+1} C_k f_k = \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} \sum_{n=-M}^M C_k e^{in(\phi-\phi_k)} \\ &= \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} C_k [1 + 2 \sum_{n=1}^M \cos n(\phi - \phi_k)] \\ &= \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} C_k + \frac{2}{2\pi(2M+1)} \left[\sum_{n=1}^M \left(\sum_{k=1}^{2M+1} C_k \cos(n\phi_k) \right) \cos(n\phi) + \sum_{n=1}^M \left(\sum_{k=1}^{2M+1} C_k \sin(n\phi_k) \right) \sin(n\phi) \right] \\ &\quad \cdots \end{aligned}$$

可以看到，要将本征函数分为偶函数或奇函数就是要求 $\sum_{k=1}^{2M+1} C_k \cos(n\phi_k)$ 或 $\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)$ 其中某一个对所有的 n 的取值均为 0，（实际计算会有一些非零误差，这是因为计算本省误差和 QR 分解算法中终止循环的判定条件也不是完全理想的对角阵，只是矩阵元素绝对值足够小。）

换言之，若想取得“偶宇称”本征函数，就要求 $\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)$ 对所有的 n 全接近于 0，这在 M 很小的时候可以很精确的满足，一般都小于 10^{-12} （比之前设置的 QR 分解终止循环的临界判定 10^{-10} 还小，可见在精度允许范围内）。但对 M 较大的情况，这种方法计算误差比较大，因此将偶宇称的判定条件改为， $\sum_{k=1}^{2M+1} C_k \cos(n\phi_k)$ 对所有的 n 绝对值最大值远大于 $\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)$ 。通过实验，发现第 0, 2, 4, 6, 8, 10 就是对应于偶宇称本征函数的本征值，可以直接通过这个规律求解。

对于第四问，我们完全可以采用上面同样的方法判定偶宇称的本征值，然后计算本征函数系数，从而求出本征函数，并在 0 到 90 度上画图即可。本征向量的求解是记录每一步 Householder 变换和 Givens 的初等矩阵，如上文中分析的，QR 分解得到的 Q 矩阵的每列就是矩阵 A 本征向量，R 矩阵的对角元就是 A 矩阵的本征值。在后文可以看到，经过验证， $\mathbf{AQ} = \lambda \mathbf{Q} < 10^{-10}$ ，可以认为在误差允许范围内，分解有效。

1.3 算法描述和实现

1.3.1 第 (1) (2) (3) 问

首先先预定义一部分工具函数，包括正负判断，求模长，求绝对值等。然后对需要分解的 $\mathbf{A}=\mathbf{T}+\mathbf{V}$ 矩阵进行初始化：

```

1 def sign1(a): # sign1就是返回正负的函数
2     if a > 0: return 1
3     if a < 0: return -1
4
5
6 def fs2(a, m): # 定义2范数函数，返回模长
7     c = 0
8     for i in range(0, m, 1):
9         c += a[i]**2
10    c = math.sqrt(c)
11    return c
12
13
14 def jdz(a): # 返回函数的绝对值
15    if a >= 0:
16        return a
17    else:
18        return -a
19

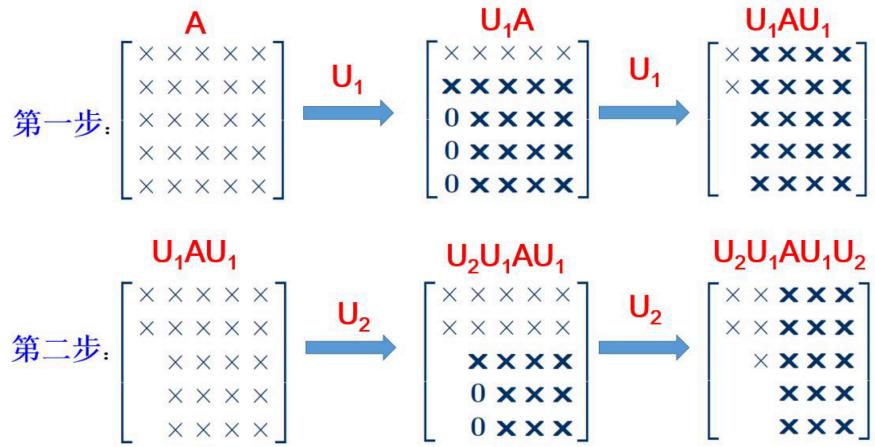
```

```

21 for q in range(q1, qr, 1): # 从q1到qr (qr取不到) , 循环求解矩阵
A = [[0 for i in range(2 * M + 2)] for i in range(2 * M + 2)]
23 for j in range(1, 2 * M + 2, 1): # 赋值
    for k in range(1, 2 * M + 2, 1):
25 if j == k:
        A[j][j] = M * (M + 1) / 3 + 2 * q * math.cos(2 * phi[j])
    ]
27 else:
        A[j][k] = ((-1)**(j - k)) * math.cos(math.pi * (j - k) / (2 * M + 1)) / (
29 2 * math.sin(math.pi * (j - k) / (2 * M + 1)) ** 2)
A = np.asarray(A)

```

接着进行 Householder 变换将 A 矩阵变为上三角矩阵，具体算法如下：



第n-2步以后:

$$\frac{U_{n-2} \cdots U_2 U_1 A U_1 U_2 \cdots U_{n-2}}{=U_0^T} = B = \begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix} \text{ (上三角矩阵)}$$

若A为对称矩阵，则对角线以上也会有0，例如第一步变为：

$$A \xrightarrow{U_1} U_1 A \xrightarrow{U_1} U_1 A U_1$$

若A为任意复数矩阵，则：

第一步：

$$\begin{array}{c} \text{A} \\ \left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{array} \right] \xrightarrow{Q_1^H} \text{Q}_1^H A \left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times \end{array} \right] \xrightarrow{Q_1} \text{Q}_1^H A Q_1 \left[\begin{array}{cccccc} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{array} \right] \end{array}$$

第n-2步以后：

$$\frac{\text{Q}_{n-2}^H \cdots \text{Q}_2^H \text{Q}_1^H A Q_1 Q_2 \cdots Q_{n-2}}{= \text{Q}_0^H} = \text{Q}_0 = \left[\begin{array}{cccc} \times & \times & \times & \times \\ \times & \times & \times & \times \end{array} \right]$$

- 算法：输入： $\mathbf{A} \in \mathcal{R}^{m \times m}$; 输出： \mathbf{A} .

For $k = 1$ to $m - 2$

$\mathbf{x} = A_{k+1:m,k}$ (当前步的列矢量)

$\mathbf{v}_k = \text{sign}(\mathbf{x}_1) \|\mathbf{x}\|_2 \mathbf{e}_1 + \mathbf{x}$ (构造反射法线矢量)

$\mathbf{v}_k = \mathbf{v}_k / \|\mathbf{v}_k\|_2$ (归一化)

$A_{k+1:m,k:m} = A_{k+1:m,k:m} - 2\mathbf{v}_k (\mathbf{v}_k^T A_{k+1:m,k:m})$

$A_{1:m,k+1:m} = A_{1:m,k+1:m} - 2(A_{1:m,k+1:m} \mathbf{v}_k) \mathbf{v}_k^T$

End

- 对于对称矩阵，上述变换将会保持其原有的对称性，因此结果为对称三对角阵。

具体实现过程中必须加入 \mathbf{U}_k 矩阵来存储对称变换的初等矩阵，在后面会用来累乘得到 \mathbf{Q} 矩阵。具体代码实现如下：

```

1   Uk = [[0 for i in range(2 * M + 1)] for i in range(2 * M + 1)] # 用来存储每一步的H反射向量，累乘即可得到本征向量
2   for i in range(0, 2 * M + 1, 1):
3       Uk[i][i] = 1.0
4   Uk = np.asarray(Uk)

6   for k in range(1, 2 * M, 1):
7       x = A[k + 1:2 * M + 2, k]
8       e1 = [0 for i in range(2 * M - k + 1)]
9       e1[0] = 1.0
10      e1 = np.asarray(e1)
11      v = sign1(x[0]) * fs2(x, 2 * M - k + 1) * e1 + x # 计算向量v
12      v = v / fs2(v, 2 * M - k + 1)
13      vvT = [[0 for i in range(2 * M - k + 1)] for i in range(2 * M - k + 1)] # vvT矩阵
14      Uk0 = [[0 for i in range(2 * M + 2)] for i in range(2 * M + 2)]

```

```

    ] # 用来存储每一步的H反射矩阵
    for i in range(1, 2 * M + 2, 1):
        16      Uk0[i][i] = 1.0
        Uk0 = np.asarray(Uk0)
        for i in range(0, 2 * M - k + 1, 1):
            18      for j in range(0, 2 * M - k + 1, 1):
                20          vvT[i][j] = v[i] * v[j] # vvT矩阵
            vvT = np.asarray(vvT)
            22      Uk0[k + 1:2 * M + 2, k + 1:2 * M + 2] = Uk0[k + 1:2 * M + 2, k
                + 1:2 * M + 2] - 2 * vvT # UK0部分赋予反射部分
            Uk0 = Uk0[1:2 * M + 2, 1:2 * M + 2]
            24      Uk = np.matmul(Uk, Uk0) # 每次累乘得到最终的H反射矩阵
            A[k + 1:2 * M + 2, k:2 * M + 2] = A[k + 1:2 * M + 2, k:2 * M +
                2] - 2 * np.matmul(vvT, A[k + 1:2 * M + 2, k:2 * M + 2])
            # H反射矩阵对A的分块矩阵左乘
            26      A[1:2 * M + 2, k + 1:2 * M + 2] = A[1:2 * M + 2, k + 1:2 * M +
                2] - 2 * np.matmul(A[1:2 * M + 2, k + 1:2 * M + 2], vvT)
            # H反射矩阵对A的分块矩阵右乘
            A = np.asarray(A[1:2 * M + 2, 1:2 * M + 2])
            28      for i in range(0, 2 * M + 1, 1):
                for j in range(0, 2 * M + 1, 1):
                    30          if jdz(A[i][j]) < 10 ** (-10):
                        A[i][j] = 0 # 原则上此时A应该是三对角矩阵，但因为计算
                            原因，会保留非常小的数，此处直接把其赋为0

```

同理，下面的算法和代码展示了嵌套原点位移的同时，对矩阵实现 Givens 旋转变换（此处引用了刘川老师的计算物理讲义）：

选择一个形式如 (6.49) 的 Givens 转动矩阵 $G(1, 2, \theta_1)^{(1)} \equiv G_1^{(1)}$ 作用于 $H^{(0)}$ ，使得 $(G_1^{(1)})^T H^{(0)}$ 的第 (2, 1) 元素为零。我们可以调整参数 θ_1 使得原先的 $H^{(0)}$ 中左上角的 2×2 矩阵 – 图中用红色标出的部分 – 的左下角矩阵元为零。这样左乘之后我们得到：

$$(G_1^{(1)})^T H^{(0)} = \begin{bmatrix} \times & \times & \times & \times & \cdots & \times \\ 0 & \times & \times & \times & \cdots & \times \\ 0 & \bullet & \bullet & \bullet & \cdots & \bullet \\ 0 & ; & \bullet & \cdots & \cdots & ; \\ 0 & ; & \cdots & \cdots & \cdots & \bullet \\ 0 & \cdots & \cdots & \cdots & \cdots & \bullet \end{bmatrix} \quad (6.52)$$

其中矩阵的前两行是与原先矩阵比较发生了变化的部分，同时我们用 \times 表示它是与原先的 \bullet 不同的（一般来说非零的）矩阵元。

选择第二个 Givens 转动矩阵 $G(2, 3, \theta_2)^{(1)} \equiv G_2^{(1)}$ 作用于上部得到的矩阵之上, 只不过它的作用的目的是上面矩阵中蓝色的 2×2 的块。这样的矩阵的形式为:

$$G(2, 3, \theta_2) = G_2^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \hat{G}(\theta_2) & 0 \\ 0 & 0 & \mathbb{1}_{(n-3) \times (n-3)} \end{bmatrix}, \quad (6.53)$$

我们可以调节 θ_2 使得原矩阵的 $(3, 2)$ 矩阵元—也就是蓝色 2×2 块矩阵的左下角矩阵元为零。这时我们得到的矩阵看起来是如下的模样:

$$\left(G_2^{(1)}\right)^T \left(G_1^{(1)}\right)^T H^{(0)} = \begin{bmatrix} \times & \times & \times & \times & \cdots & \times \\ 0 & \blacksquare & \blacksquare & \blacksquare & \cdots & \blacksquare \\ 0 & 0 & \blacksquare & \blacksquare & \cdots & \blacksquare \\ 0 & \vdots & \bullet & \cdots & \cdots & \vdots \\ 0 & \vdots & \cdots & \cdots & \cdots & \bullet \\ 0 & \cdots & \cdots & \cdots & \cdots & \bullet \end{bmatrix} \quad (6.54)$$

继续上述步骤, 我们用 $n-1$ 个 Givens 转动获得矩阵 $\left(G_{n-1}^{(1)}\right)^T \cdots \left(G_1^{(1)}\right)^T H^{(0)}$, 我们调节每个 Givens 矩阵的转角使得最终的矩阵变为上三角矩阵:

$$(Q^{(1)})^T H^{(0)} \equiv \left(G_{n-1}^{(1)}\right)^T \cdots \left(G_1^{(1)}\right)^T H^{(0)} = R^{(1)}. \quad (6.55)$$

其中 $R^{(1)}$ 是一个上三角矩阵。

实对称阵的实用QR算法

输入: n, \mathbf{A} ; 输出: $\lambda_1, \dots, \lambda_n$.

利用Householder-Hessenberg约化将 \mathbf{A} 化为对称三对角阵;

$k := n$ (对前 k 行、 k 列执行QR算法)

while ($k > 1$) do

 if $a_{k,k-1} = 0$ then

$k := k - 1$;

 End

$s := a_{kk}$;

 For $j = 1, 2, \dots, k$

$a_{jj} := a_{jj} - s$;

 End

实对称阵的实用QR算法

用Givens旋转变换将 $\mathbf{A}_{1:k,1:k}$ 化为上三角阵，然后

计算旋转阵 \mathbf{G}_j ($j = 1, \dots, k$)的参数 c_j, s_j ；

For $j = 1, \dots, k - 1$

$\mathbf{A}_{1:k,1:k} = \mathbf{A}_{1:k,1:k} \mathbf{G}_j^T$;

End

For $j = 1, \dots, k$

$a_{jj} := a_{jj} + s$;

end

End

最终， \mathbf{A} 的对角元素就是待求的特征值： $\lambda_1, \lambda_2, \dots, \lambda_n$ 。

在下面的代码实现中，同 Household 变换一样，利用矩阵 GT 存储了 Givens 变换中的旋转变换初等阵，从而累乘得到 Q。如上文所述， $Q = U_1 U_2 \dots U_{n-2} G_1^{-1} G_2^{-1} \dots G_{n-1}^{-1}$ ，在代码的迭代过程中， $Q = \text{np.matmul}(U_k, GT)$ 。具体代码如下：

```
1 GT = [[0 for i in range(n)] for i in range(n)] # Givens矩阵，下面累乘
2   就能获得最终Givens矩阵
3   for i in range(0, n, 1):
4     GT[i][i] = 1.0
5   GT = np.asarray(GT)
6   k = 2 * M
7   while k > 0:
8     if jdz(A[k][k - 1]) < 10 ** (-12):
9       k = k - 1
10      s = A[k][k]
11      for j in range(0, k + 1, 1):
12        A[j][j] -= s # 原点位移
13      for i in range(1, k + 1, 1):
14        a = math.sqrt(A[i - 1][i - 1] ** 2 + A[i][i - 1] ** 2)
15        c = A[i - 1][i - 1] / a
16        s1 = A[i][i - 1] / a
17        G = [[0 for i in range(k + 1)] for i in range(k + 1)]
18        for j in range(0, k + 1, 1):
19          G[j][j] = 1.0
20          G[i][i] = G[i - 1][i - 1] = A[i - 1][i - 1] / a
21          G[i][i - 1] = -s1
22          G[i - 1][i] = s1 # 单个Givens矩阵赋值
23          G = np.asarray(G)
24          A[0:k + 1, 0:k + 1] = np.matmul(G, A[0:k + 1, 0:k + 1]) # Givens矩阵左乘A的分块矩阵
25          Gni = G
26          Gni[i][i - 1] = s1
27          Gni[i - 1][i] = -s1
```

```

27      Gni = np.asarray(Gni) # 单个Givens矩阵的逆
28      A[0:k + 1, 0:k + 1] = np.matmul(A[0:k + 1, 0:k + 1], Gni)
29          # Givens矩阵的逆右乘A的分块矩阵
30      GT0 = [[0 for i in range(n)] for i in range(n)] # 每步的
31          全块的Givens矩阵
32      for i in range(0, n, 1):
33          GT0[i][i] = 1.0
34      GT0 = np.asarray(GT0)
35      GT0[0:k + 1, 0:k + 1] = Gni
36      GT = np.matmul(GT, GT0) # 累乘每步的Givens矩阵，获得最终
37          的Givens矩阵
38
39      for j in range(0, k + 1, 1):
40          A[j][j] += s # 原点位移复原
41      Q = np.matmul(Uk, GT) # 获得最终Q矩阵，Q矩阵每列对应A每列本征值的
42          本征向量

```

最后，为解决题中所提出的问题。考虑对任意给定的 q （可以根据需要设定为一个范围内取多个值，也可以取单一的值）和 M （由外部输入），将所得本征值（上三角矩阵对角元）和本征函数（ $\mathbf{A}=\mathbf{T}+\mathbf{V}$ 矩阵本征矢，即 Q 矩阵列向量）存储，打印并画图即可。对于偶宇称解，只需要选择第 0, 2, 4, 6, 8, 10 个本征值即可。其结果展示在下一部分中。

1.3.2 第 (4) 问

对于第 (4) 问，需要额外处理的部分就是求解关于角度的本征矢 $\Phi_i^{even}(\phi)$ 。由于 $M=50$ 较大，在这种情况下情况，偶宇称的判定条件设定为为， $\sum_{k=1}^{2M+1} C_k \cos(n\phi_k)$ 对所有的 n 绝对值最大值远大于 $\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)$ 。根据此前 QR 分解法求出的 $\mathbf{A}=\mathbf{T}+\mathbf{V}$ 矩阵本征矢，即 Q 矩阵的列向量，我们可以轻松得到对于不同的本征值下的 $\Phi(\phi) = \sum_{k=1}^{2M+1} C_k f_k(\phi)$ （其中在规定基矢坐标下，本征矢的每一个元素就对应与 C_k ）。根据展开式：

$$\begin{aligned}
\Phi(\phi) &= \sum_{k=1}^{2M+1} C_k f_k = \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} \sum_{n=-M}^M C_k e^{in(\phi-\phi_k)} \\
&= \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} C_k [1 + 2 \sum_{n=1}^M \cos n(\phi - \phi_k)] \\
&= \frac{1}{2\pi(2M+1)} \sum_{k=1}^{2M+1} C_k + \frac{2}{2\pi(2M+1)} [\sum_{n=1}^M (\sum_{k=1}^{2M+1} C_k \cos(n\phi_k)) \cos(n\phi) + \sum_{n=1}^M (\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)) \sin(n\phi)]
\end{aligned}$$

.....

我们就可以求出关于角度的偶宇称本征矢，整个过程包括了将本征值从小到大的排序存储（这样就可以得到题中所要求的头 6 个偶宇称的本征值），也包括了对 QR 算法有效和本征值有效的检验，具体结果会展示在下一部分，具体代码实现如下所示，：

```

1   B = [0 for i in range(2 * M + 1)] # c存储按照从小到大排的本征值
2   pp = [i for i in range(2 * M + 1)] # 存储本征值变换的顺序
3   for i in range(0, 2 * M + 1, 1):
4       B[i] = A[i][i]
5
6
7   for i in range(0, 2 * M + 1, 1):
8       for j in range(i + 1, 2 * M + 1, 1):
9           if B[j] < B[i]: # 按照从小到大排序
10              sxs = pp[i]
11              pp[i] = pp[j]
12              pp[j] = sxs
13              cc = B[j]
14              B[j] = B[i]
15              B[i] = cc
16
17
18   sd = 0
19   for l in range(0, n, 1): # 按顺序对所有本征值遍历
20
21       maxdf = 0
22       maxddf = 0
23       #print(B[l])
24       #print(l)
25       for i in range(1, M + 1, 1):
26           df = 0
27           ddf = 0
28           for k in range(0, n, 1):
29               df += Q[k][pp[1]] * math.sin(i * phi[k + 1]) #
30               ddf += Q[k][pp[1]] * math.cos(i * phi[k + 1])
31
32           if maxdf < jdz(df):
33               maxdf = jdz(df)
34           if maxddf < jdz(ddf):
35               maxddf = jdz(ddf)
36
37       if maxddf == 0:
38           continue
39
40       ggg = maxdf / maxddf
41       if jdz(ggg) < 0.01: # 对任何n, 此时sin函数前面系数均为0, 即为
42           偶函数
43           pxl[sd][q - ql] = B[l] # 存本征值
44           bzxl[sd][q - ql] = pp[1]
45           sd += 1
46       #print("haha")

```

```

        if sd == 6:
47            break
48            # print(sd)

49

51 jiaodu = [0.0 for i in range(101)] # 0到90度取100个数
52 for i in range(0, 101, 1):
53     jiaodu[i] = i * math.pi / 200 # 每个数赋值
54 jiaodu = np.asarray(jiaodu)
55 for i in range(0, 6, 1):
56     wz = bzx1[i][0] # wz就是本征向量在Q的列数
57     y = [0.0 for j in range(101)]
58     y = np.asarray(y) # y就是每个本征值下的解函数
59     for nn in range(1, M + 1, 1):
60         osx = 0.0
61         jsx = 0.0
62         for k in range(0, n, 1):
63             osx += Q[k][wz] * math.cos(nn * phi[k + 1]) # 用来计算每
64             个cos函数前的系数
65             jsx += Q[k][wz] * math.sin(nn * phi[k + 1]) # 用来计算每
66             个sin函数前的系数
67             y += osx * np.cos(nn * jiaodu) + jsx * np.sin(nn * jiaodu) #
68             角度相关赋予
69             #print(osx)
70             #print(jsx)
71             y *= 2
72             for k in range(0, n, 1):
73                 y += Q[k][wz] # 加上常数项
74             # y=y/(2*math.pi*n)
75             if y[1] < 0: y = -y
76             plt.plot(jiaodu, y, label=pxl[i][0])
77             plt.legend()
78             show()

79 #检验QR算法和本征值
80 for q in range(q1, qr, 1):
81     A = [[0 for i in range(2 * M + 2)] for i in range(2 * M + 2)]
82     for j in range(1, 2 * M + 2, 1):
83         for k in range(1, 2 * M + 2, 1):
84             if j == k:
85                 A[j][j] = M * (M + 1) / 3 + 2 * q * math.cos(2 * phi[j])
86             else:
87                 A[j][k] = ((-1) ** (j - k)) * math.cos(math.pi * (j -
88                     k) / (2 * M + 1)) / (
89                     2 * math.sin(math.pi * (j - k) / (2 * M + 1)) ** 2)
90
91 A = np.asarray(A)

```

```
89 for i in range(n):
    print(np.matmul(A[1:n+1,1:n+1],Q[:,pp[i]])-B[i]*Q[:,pp[i]]))
```

1.4 结果展示

1.4.1 (1)

取 $M=45$, $q=8$, 求出本征值与老师的参考进行对比, 发现符合的非常好。

```
-10.60672918499967
-10.605367683292016
-0.43594356680390484
-0.38936176536441713
8.115238831186085
8.709914363649247
14.181880362423879
17.18252778259867
19.252705060936034
26.220999473538534
26.577753313794908
```

图 1.1: 程序结果

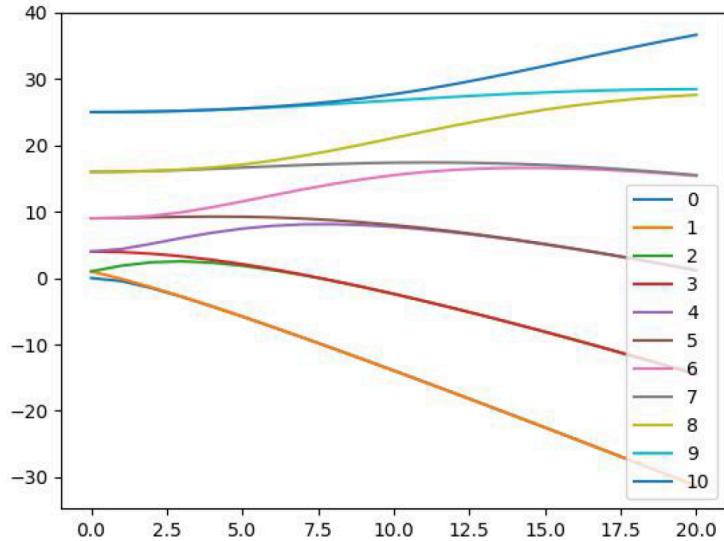
为了你们的方便, 我列出当 $q = 8.0$ 、 $M = 45$ 时, 我得到的头 11 个本征值:

```
1 th : -10.6067292355529
2 th : -10.6053681387929
3 th : -0.435943601321071
4 th : -0.389361770181921
5 th : 8.11523883026308
6 th : 8.70991435760512
7 th : 14.1818803623163
8 th : 17.1825277707890
9 th : 19.2527050594243
10 th : 26.2209994726556
11 th : 26.5777532926020
```

图 1.2: 老师给出的结果参考

1.4.2 (2)

取 $M=50$, $q=0$ 到 20, 将头 11 个本征值（排序后的本征值）做图如下。横坐标是 q , 纵坐标是本征值。



1.4.3 (3)

为验证偶宇称解筛选规律（即第 0、2、4、6、8、10 个本征值即为偶宇称本征矢对应本征值），选取 $M=5$, $q=10$ 循环输出 $\sum_{k=1}^{2M+1} C_k \cos(n\phi_k)$ 和 $\sum_{k=1}^{2M+1} C_k \sin(n\phi_k)$, 发现二者相差十分悬殊。（ \cos 系数, 10^0 量级, 远大于 \sin 系数, 10^{-12} ）。可见对于一个确定的本征向量, \cos 和 \sin 代表的本征函数是可以很好地被区分开来的。具体结果如下：

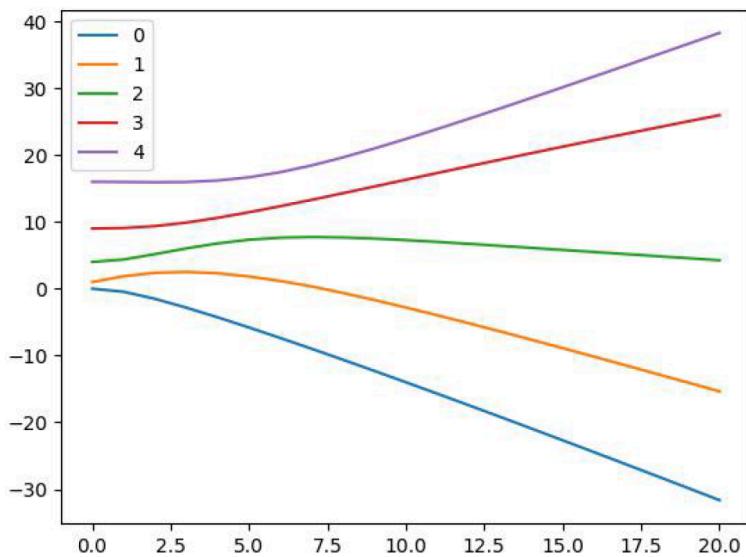
```

请输入M值: 5
0.03675790723718309
-7.981830801967961e-12
1.5904764291676068
2.585425316160535e-13
-0.09186856105340206
3.969614312719985e-12
-0.5884567538399885
-4.657329304234089e-13
0.17447428427137487
-1.138577711199047e-12
1.2748954797328973
6.609963148720019e-14
-0.05641924784927684
-2.958798728910054e-14
-1.7463039962527116
1.732145036214406e-14
-0.38050082610366026
-1.0873656802346748e-18
0.7678654167614799
-1.9391415476868882e-14
-0.7608049663278854
-1.4112356775904752e-13
0.6376342057642514
1.279802720809003e-13

```

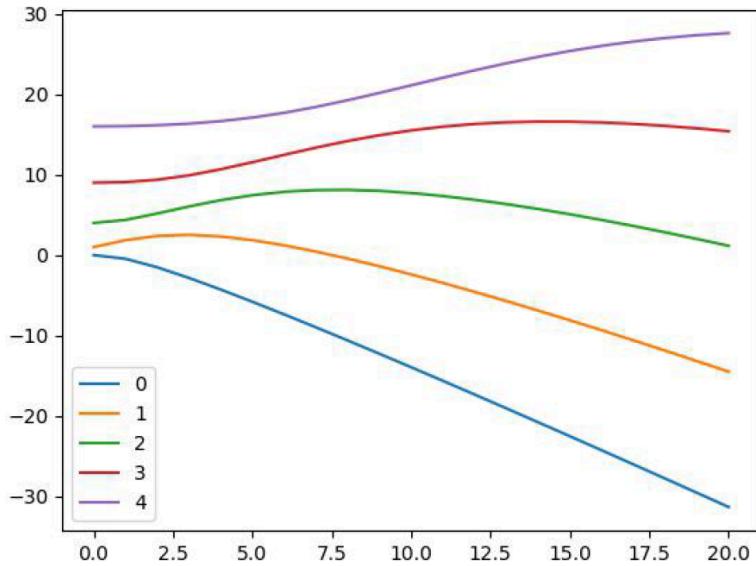
图 1.3: 循环输出 cos 和 sin 系数

对于 M=5, 头五个偶宇称本征值做图如下:



对于 M=40, 上述方法误差比较大, 因此我选择用两者最大之比作为评判标准。但显

然直接用之前发现的规律计算，可以大大缩减时间，所以我们直接用规律进行计算。至于这个计算的正确性，在下一问， $M=50$ 时做出的图形显然可以作为一个不完全的例子来证明它是有效的。 $M=40$ 的结果如下图所示：



1.4.4 (4)

这里利用以下代码验证本征值和本征向量的有效性 (\mathbf{Q} 矩阵分解正确)，即验证上文所述 $\mathbf{AQ} = \lambda \mathbf{Q} < 10^{-10}$ 。下面结果以 $M=5, q=10$ 为例，经验证， $M=50$ 也同理有效。

```

1  for q in range(q1, qr, 1):
2      A = [[0 for i in range(2 * M + 2)] for i in range(2 * M + 2)]
3      for j in range(1, 2 * M + 2, 1):
4          for k in range(1, 2 * M + 2, 1):
5              if j == k:
6                  A[j][j] = M * (M + 1) / 3 + 2 * q * math.cos(2 * phi[j])
7              else:
8                  A[j][k] = ((-1)**(j - k)) * math.cos(math.pi * (j -
9                      k) / (2 * M + 1)) / (
10                     2 * math.sin(math.pi * (j - k) / (2 * M + 1)) ** 2)
11
12      A = np.asarray(A)
13
14      for i in range(n):
15          print(np.matmul(A[1:n+1, 1:n+1], Q[:, pp[i]]) - B[i]*Q[:, pp[i]])

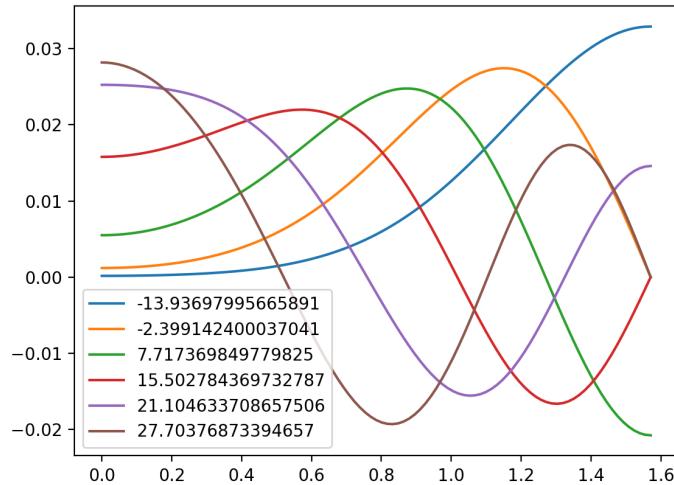
```

```

[-4.72955008e-14 -1.70530257e-13 2.13162821e-14 5.01820807e-14
 5.77315973e-15 1.30312428e-14 6.57252031e-14 1.10134124e-13
-1.15463195e-13 -3.96349620e-14 3.94823063e-15]
[-4.27435864e-14 -1.05693232e-13 1.06581410e-13 7.63833441e-14
 6.41153797e-15 6.36990460e-15 7.32747196e-14 1.08357767e-13
-1.14575016e-13 -3.81916720e-14 -1.09720586e-14]
[-2.88657986e-15 5.10702591e-15 -1.33226763e-15 4.44089210e-16
 2.33146835e-15 -3.19189120e-16 6.77236045e-15 1.44328993e-15
 6.43929354e-15 1.66533454e-16 4.23559690e-15]
[-9.65894031e-15 4.21884749e-15 -2.55351296e-15 -5.32907052e-15
-9.04831765e-15 -1.11369247e-13 -2.45470311e-13 4.09450251e-13
 9.41469125e-14 -6.15063556e-14 -9.09688991e-15]
[-6.19504448e-14 -1.64201985e-13 8.97060204e-14 9.76996262e-15
-5.32907052e-15 2.59792188e-14 1.30562228e-13 5.28466160e-14
-1.40443213e-13 -3.81916720e-14 -3.66373598e-15]
[ 3.30846461e-14 1.30673250e-13 -4.84057239e-14 -1.22568622e-13
-2.73114864e-14 -3.77475828e-14 -1.19904087e-13 -4.81836793e-14
 1.36890499e-13 3.66373598e-14 -2.63369599e-16]
[-5.32907052e-15 -4.44089210e-16 2.66453526e-15 9.65894031e-15
-2.66453526e-15 2.22044605e-15 -4.32986980e-15 5.10702591e-15
-7.54951657e-15 1.42108547e-14 3.55271368e-15]
[ 1.24344979e-14 4.44089210e-16 3.33066907e-15 7.54951657e-15
-1.77635684e-15 2.44249065e-15 -2.22044605e-15 1.77635684e-15
-2.22044605e-15 1.77635684e-15 1.12525071e-14]
[-9.99200722e-16 4.44089210e-16 4.66293670e-15 7.99360578e-15
-2.13162821e-14 -5.32907052e-15 -8.88178420e-16 -2.66453526e-15
 9.99200722e-15 1.33226763e-15 -4.44089210e-15]
[-5.32907052e-15 -2.22044605e-15 8.43769499e-15 6.21724894e-15
-2.84217094e-14 7.10542736e-15 5.32907052e-15 2.66453526e-15
-3.55271368e-15 5.55111512e-15 -2.33554630e-15]
[ 1.24344979e-14 -4.44089210e-16 -4.44089210e-15 2.88657986e-15
-2.10942375e-15 1.11022302e-16 2.22044605e-16 1.11022302e-15
-2.22044605e-15 3.55271368e-15 -7.10542736e-15]

```

接着将本题所求结果展示如下。其中，由于本征矢量加个负号仍然是本征矢量，为了和答案给的参考图形一致，此处给几条线加个负号。同理，本征向量乘以一个系数也是本征向量，所以纵坐标的具体值也不是关心的重点。如下图所示，结果良好：



Chapter 2

第二题

2.1 题目

(一) 【30分】本题中, 我们来考虑一个氢原子在强激光场下电离的物理问题。在外加激光场的电场的偶极作用下(即在电子运动的范围内不考虑激光场电场的空间依赖), 一个电子从氢原子的基态 $|\Psi_0(t)\rangle$ 跃迁至自由电子在激光场中的本征态 $|\Psi_p^V(t)\rangle$ (即Volkov态)并获得末态动量 \mathbf{p} 。根据量子力学, 这个过程的跃迁几率幅为:

$$M_p(t_f, t_i) = \langle \Psi_p^V(t_f) | U(t_f, t_i) | \Psi_0(t_i) \rangle, \quad (1)$$

其中:

$$|\Psi_0(t)\rangle = e^{iI_p t} |\Psi_0\rangle, \quad I_p \text{ 为氢原子基态的电离势}, \quad (2)$$

$$|\Psi_p^V(t)\rangle = |\mathbf{p} + \mathbf{A}(t)\rangle e^{-iS_p(t)}, \quad (3)$$

$$S_p(t) = \frac{1}{2} \int_0^t d\tau [\mathbf{p} + \mathbf{A}(\tau)]^2, \quad (4)$$

公式(1)中, $U(t_f, t_i)$ 是系统整体哈密顿量 $H(t)$ 的时间演化算符:

$$H(t) = -\frac{1}{2}\nabla^2 + \mathbf{r} \cdot \mathbf{E}(t) + V(\mathbf{r}), \quad (5)$$

其中 $\mathbf{r} \cdot \mathbf{E}(t)$ 表示电子与外加光场的偶极相互作用, $V(\mathbf{r})$ 表示电子受到原子核的库仑势。因为演化算符 $U(t_f, t_i)$ 满足Dyson方程, 则可将 $M_p(t_f, t_i)$ 展开为:

$$\begin{aligned} M_p(t_f, t_i) &= -i \int_{t_i}^{t_f} dt \langle \Psi_p^V(t_f) | U_f(t_f, t) \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0(t) \rangle \\ &\quad - \int_{t_i}^{t_f} dt \int_t^{t_f} d\tau \langle \Psi_p^V(t_f) | U_f(t_f, t) V(\mathbf{r}) U(\tau, t) \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0(t) \rangle. \end{aligned} \quad (6)$$

在一定条件下, 可以仅考虑上式中右边第一项的贡献:

$$\begin{aligned} M_p^0(t_f, t_i) &= -i \int_{t_i}^{t_f} dt \langle \Psi_p^V(t_f) | U_f(t_f, t) \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0(t) \rangle, \\ &= -i \int_{t_i}^{t_f} dt \int d^3k \langle \Psi_p^V(t_f) | \Psi_k^V(t_f) \rangle \langle \Psi_k^V(t_f) | \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0(t) \rangle, \\ &= -i \int_{t_i}^{t_f} dt \langle \Psi_p^V(t) | \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0(t) \rangle. \end{aligned} \quad (7)$$

从上式可以看出, $M_p^0(t_f, t_i)$ 的物理意义可理解为初始 t_i 时刻被束缚在氢原子基态的电子在 t 时刻受到

外加光场 $\mathbf{E}(t)$ 的作用跃迁至Volkov态并获得末态动量 \mathbf{p} 的几率:

$$M_{\mathbf{p}}^0(t_f, t_i) = -i \int_{t_i}^{t_f} dt \langle \mathbf{p} + \mathbf{A}(\mathbf{t}) | \mathbf{r} \cdot \mathbf{E}(t) | \Psi_0 \rangle e^{iS(t)}, \quad (8)$$

$$S(t) = \int_0^t d\tau \left[\frac{1}{2} (\mathbf{p} + \mathbf{A}(\tau))^2 + I_p \right]. \quad (9)$$

记 $\mathbf{q} = \mathbf{p} + \mathbf{A}(\mathbf{t})$, 则由(8)式可得:

$$M_{\mathbf{p}}^0(t_f, t_i) = \frac{1}{(2\pi)^{3/2}} \int_{t_i}^{t_f} dt \left\{ \frac{\partial}{\partial \mathbf{q}} \left[\int d\mathbf{r} e^{-i\mathbf{q} \cdot \mathbf{r}} \Psi_0(\mathbf{r}) \right] \right\} \cdot \mathbf{E}(t) e^{iS(t)}. \quad (10)$$

已知氢原子基态波函数 $\Psi_0(r) = \frac{(2I_p)^{\frac{3}{4}}}{\sqrt{\pi}} e^{-\sqrt{2I_p}r}$, 代入(10)式得到电离几率幅的直接积分(DI)表达式:

$$\begin{aligned} M_{\mathbf{p}}^0(t_f, t_i)_{DI} &= \int_{t_i}^{t_f} dt \left\{ \frac{\partial}{\partial \mathbf{q}} \left[\frac{2^{\frac{3}{2}}(2I_p)^{\frac{5}{4}}}{\pi(\mathbf{q}^2 + 2I_p)^2} \right] \right\} \cdot \mathbf{E}(t) e^{iS(t)} \\ &= 2^{\frac{7}{2}}(2I_p)^{\frac{5}{4}} \int_{t_i}^{t_f} dt \frac{\mathbf{q} \cdot \mathbf{E}(t)}{\pi(\mathbf{q}^2 + 2I_p)^3} e^{iS(t)}. \end{aligned} \quad (11)$$

对于(11)式, 可采取我们课堂上讲到的鞍点近似方法进行化简, 即给定动量 \mathbf{p} , 则满足鞍点方程

$$\frac{\partial S(t)}{\partial t} = 0, \quad t = t_s = t_r + it_i, \quad (12)$$

由(11)式可得到鞍点近似下(SPM)的电离几率幅表达式:

$$M_{\mathbf{p}}^0(t_f, t_i)_{SPM} = -\frac{(2I_p)^{\frac{5}{4}}}{\sqrt{2}} \sum_{\alpha} \frac{1}{S''(t_{s\alpha})} e^{iS(t_{s\alpha})}, \quad (13)$$

$$\text{其中: } S''(t_{s\alpha}) = \left. \frac{\partial^2 S}{\partial t^2} \right|_{t=t_{s\alpha}}. \quad (14)$$

注意此问题中, 上面的对所有鞍点 $t_{s\alpha}$ 的求和中, 所有虚部小于0的鞍点($t_{s\alpha} < 0$)非物理, 请舍去。

前面的铺垫你可以选择不太刻意在乎, 现在请严肃考虑给你们的计算问题。考虑一个脉冲长度仅为两个周期、偏振方向沿着 \mathbf{e}_z 的外加激光场, 其矢势 $\mathbf{A}(\mathbf{t})$, 电场 $\mathbf{E}(t)$ 以及电场强度 E_0 分别为:

$$\mathbf{A}(\mathbf{t}) = A_0 \sin(\omega t) \sin^2\left(\frac{\omega t}{4}\right) \mathbf{e}_z, \quad (15)$$

$$\mathbf{E}(\mathbf{t}) = -\frac{\partial \mathbf{A}(\mathbf{t})}{\partial t}, \quad (16)$$

$$E_0 = A_0 \omega. \quad (17)$$

波长 $\lambda = 3200nm$, 峰值光强 $I_0 = 5 \times 10^{13} W/cm^2$, 氢原子电离势 $I_p = 13.6 \text{ eV}$, ω 为激光频率; 起始时刻 $t_i = 0$, 结束时刻 $t_f = \frac{4\pi}{\omega}$; 不考虑垂直于激光偏振方向上的电子末态动量, 即取 $p_{x,y} = 0$ (a.u.),

仅考虑 \mathbf{e}_z 方向如下范围的电子 $\mathbf{p} = p_z \mathbf{e}_z = 0 \sim 2$ (a.u.); 因此, 电子末态能量 $E_k = \frac{1}{2} p_z^2$.

- (1) 请选择合适的非线性方程求根方法, 求出 $p_z = 1$ (a.u.)时鞍点方程 $\frac{\partial S(t)}{\partial t} = 0$ 的6个解 $t_{s1} \sim t_{s6}$;
- (2) 在 $p_z \in [0.01, 2]$ 区间上, 取间隔 $\Delta p_z = 0.2$, 给出鞍点近似(即(13)式)得到的电子能谱, 即 $|M_{\mathbf{p}}^0(t_f, t_i)_{SPM}|^2$ 与 E_k 的关系曲线图;
- (3) 在与(2)相同的区间和间隔下, 利用直接积分的方法(即(11)式)得到的电子能谱, 即 $|M_{\mathbf{p}}^0(t_f, t_i)_{DI}|^2$ 与 E_k 的关系曲线图。请画在与(2)结果的同一张图进行比较。

【注意, 你的两条曲线不会完全重合, 二者间有一个大小在(1,2)间的整体因子。但是, 除此之外, 其它细节应当一样。】

提示: 利用所学算法求解 $p_{x,y} = 0$ (a.u.), $p_z = 0 \sim 2$ (a.u.)时的鞍点方程 $\frac{\partial S(t)}{\partial t} = 0$, 每一个末态动量 p_z 对应6个物理的鞍点(注意, 虚部小于0的非物理的鞍点已舍去), 将6个解 $t_{s1} \sim t_{s6}$ (解为复数)代入式(13)得到鞍点近似下动量为 p_z 的电离几率幅; 对式(11)式进行数值积分得到直接积分的电离几率幅。(要求所有计算采用原子单位a.u., 附录里给出了相应转换关系。)

2.2 分析与思路

2.2.1 第一问 (1)

按照题目的要求求解 $\frac{\partial S}{\partial t} = 0$ 即可，对式子进行化简：

$$S(t) = \int_0^t d\tau \left(\frac{1}{2}(p + A(\tau))^2 + I_p \right)$$

所以

$$\frac{\partial S}{\partial t} = \frac{1}{2}(p + A(t))^2 + I_p = 0$$

解这个方程即可这是一个周期性的复数方程，可以用课上提到的非线性方程解法解，一开始我使用了牛顿法，但是收敛性不好，所以来采用了弦切法，即：

对于给定寻根区间 (a, b) ，选取

$$q_k = q = \frac{f(b) - f(a)}{(b - a)}, \quad \forall k \geq 0.$$

然后，给定一个初值 x_0 ，做如下迭代：

$$x^{(k+1)} = x^{(k)} - \frac{(b - a)}{f(b) - f(a)} f(x^{(k)}), \quad k \geq 0.$$

具体来说，使用弦切法永远需要两个端点值，每次循环这两个端点值不变，同时用一个随机数产生器在端点内产生一个初始值，然后进行解的迭代。端点的取法是这样的：把解空间分割成 200×60 份，实部在 0 到 $\frac{4\pi}{\omega}$ 之间分 200 份，然后虚部在 0 到 150 之间分 60 份，依次寻找解。

同时经过尝试可知，一般情况下在同一个区间内迭代 100 次就能达到所需的精度 10^{-12} 次方，为了避免陷入死循环，必须设置迭代超过 100 次就退出重新设初始值。同时，为了避免指数溢出，必须加一旦溢出就退出循环再找随机数，另外，为了避免找到相同的解，加一个判断函数，如果相同，将不把解加入解向量里，重新找新的解，直到有 6 个符合要求的解被找到。

2.2.2 第二问 (2) & 第三问 (3)

(2) 鞍点近似下的电离几率幅表达式:

$$M_p^0(t_f, t_i)_{\text{saddlepointmethod}} = -\frac{(2I_p)^{\frac{5}{4}}}{\sqrt{2}} \sum_{\alpha} \frac{1}{S''(t_{s\alpha})} e^{iS(t_{s\alpha})}$$

对鞍点求和即可。

其中的一些量表示如下:

$$S''(t_{s\alpha}) = \frac{\partial^2 S}{\partial t^2}|_{t=t_{s\alpha}}$$

$$S(t) = \int_0^t d\tau \left(\frac{1}{2}(p + A(\tau))^2 + I_p \right)$$

将 $S''(t_{s\alpha})$ 求出:

$$S''(t_{s\alpha}) = \omega A_0 \cos(\omega t) \sin^2(\omega t/4) + \frac{\omega}{4} \sin(\omega t) \sin(\omega t/2)$$

可以论证, 鞍点在本题的背景下只能有6个有物理意义的解, 因为:

$$A(t) = A_0 \sin(\omega t) \sin^2\left(\frac{\omega t}{4}\right) = A_0 \sin(\omega t/2) \cos(\omega t/2) (1 - 2\cos^2(\omega t/2))$$

令 $\cos(\omega t/2) = x$

$$\left(\frac{1}{2}(p + A(t))^2 + I_p\right) = \left(\frac{1}{2}(p + A_0 \sqrt{1-x^2} * x * (1-2x^2))^2 + I_p\right)$$

可以转化为6次多项式求根问题。有六个符合物理的解。对 E_k 取一些点, 得到相应的 P_z 算出点然后再进行求和即可。

(3) 积分下的电离几率幅表达式:

$$M_p^0(t_f, t_i)_{\text{directintegrate}} = 2^{\frac{7}{2}} (2I_p)^{\frac{5}{4}} \int_{t_i}^{t_f} dt \frac{qE(t)}{\pi(q^2 + 2I_p)^3} e^{iS(t)}$$

其中:

$$q = p + A(t)$$

$$E(t) = -\frac{\partial A(t)}{\partial t}$$

将 q 和 E 算出积分即可, 由于这两问有很多相同步骤, 所以放在一个程序里最后一起画图。

此处需要说明的是, 对第二问鞍点求和法而言, 在具体的实现过程中, 考虑到这是对 200 个 p 值而言, 每一个 p 值下求 6 个解, 为减少运算量, 此处选择先对 $p=0$ 用第一问的方法求出 6 个解, 再在原有解空间附近范围寻找其他 p 值下的解, 这是因为对于相邻参数 p , 解是十分接近的, 直接利用上一步的解可以在少量的迭代步数下得到较为精确的解。最后求 $M_p^0(t_f, t_i)_{\text{SPM}}$ 时, 对 6 个解求和得到。

至于第三问积分, 首先是将 $S(t)$ 积分, 然后对 $M_p^0(t_f, t_i)_{\text{DI}}$ 再积分, 然后求模方, 最后以 E_k 为横坐标画图。这里所有的积分过程均选用的是复化抛物线积分法, 具体积分算法如下:

- 记区间 $[x_k, x_{k+1}]$ 的中点 $(x_k + x_{k+1})/2$ 为 $x_{k+\frac{1}{2}}$, 则复化抛物线求积公式为:

$$S_n = \sum_{k=0}^{n-1} \frac{h}{6} [f(x_k) + 4f(x_{k+\frac{1}{2}}) + f(x_{k+1})] \quad (109)$$

$$\begin{aligned} &= \frac{h}{6} \left[f(x_0) + 4f(x_{\frac{1}{2}}) + f(x_1) + f(x_1) + 4f(x_{1+\frac{1}{2}}) + f(x_2) \right. \\ &\quad \left. + f(x_2) + 4f(x_{2+\frac{1}{2}}) + f(x_3) + \cdots + f(x_{n-1}) + 4f(x_{n-\frac{1}{2}}) + f(x_n) \right] \\ &= \frac{h}{6} \left[f(a) + 4 \sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) + 2 \sum_{k=1}^{n-1} f(x_k) + f(b) \right] \\ &= \frac{h}{6} \left\{ f(a) + \sum_{k=1}^n [4f(x_{k-\frac{1}{2}}) + 2f(x_k)] - f(b) \right\}. \end{aligned} \quad (110)$$

2.3 算法描述和实现

2.3.1 预先定义的函数

首先需要预选定义的是解数理方程中所需要的基本函数: exp 函数对应于 e^{ix} , csin 和 ccos 对应于变量为复数的三角函数。其次, 要被定义的是求积分过程中的 S(t) 函数, 二阶导数 Sdd 函数等。此外, 我们还需要辅助的复化抛物求积分函数 mjifen(t, p), 和被积函数的数学表达 mdi(t, p) 等。最后, 为了保证我们的解不一样, 定义检索函数 jiansuo 来检查新的解是否和原有的解并不相同。具体代码实现如下:

```

1 import random
2 import math
3 import matplotlib.pyplot as plt
4
5 I0 = 5 * 10 ** (13)
6 E0 = math.sqrt(I0 / (3.5094448314 * 10 ** (16)))
7 w = 45.5633525316 / 3200
8 Ip = 0.5
9 A0 = E0 / w
10 jie = [[0 for i in range(6)] for i in range(200)] # 存放解的矩阵
11 p = [(i + 1) * 0.01 for i in range(200)] # 可调参数p
12 Ek = [0 for i in range(200)]
13 for i in range(0, 200, 1):
14     Ek[i] = p[i] ** 2 / 2 # 横坐标
15 tf = 4 * math.pi / w
16 Nx = 100
17 ti = 150
18 Ny = 30
19 MSPM = [0.0 for i in range(200)]
20 MDI = [0.0 for i in range(200)]
21
22 # SS=[[0 for i in range(10000)] for i in range(200)]
23
```

```

25
def exp(x):
    return complex(math.cos(x.real), math.sin(x.real)) * float(math.
        exp(-x.imag)) # 数学上exp(ix)

29
def csin(x): # sinx,x为复数
    return (exp(x) - exp(-x)) / 2j

33
def ccos(x): # 复数的余弦函数
    return (exp(x) + exp(-x)) / 2

37
def f(t, i): # 就是本题中的x
    y = A0 * csin(w * t) * csin(w * t / (4 + 0j)) ** 2
    y = p[i] + y # p可变
    y = 0.5 * y ** 2 + 0.5
    return y

43

45 def jiansuo(x, l, n): # 用来检索,如果两个解相同,就返回1,不相同,返
    k = 0
    for i in range(0, n, 1):
        a = x.real - jie[1][i].real # 分别对比实部和虚部
        b = x.imag - jie[1][i].imag
        if a < 10 ** (-3) and b < 10 ** (-3):
            k = 1
            break
    return k

55
def fjifen(t, p): # 用来存要积分的方程,也就是第一问为0的方程
    return (p + A(t)) ** 2 / 2 + Ip # 为时间t, 和参数pz的二元函数

59
def S(t, p): # 题目中的S函数,是第一问函数,也即f的积分,是t和pz的二
    元函数
    dt = t / 1000 # 分成1000份
    a = 0
    for i in range(1, 1001, 1):
        a += 4 * fjifen((i - 0.5) * dt, p) + 2 * fjifen(i * dt, p) #
            采用复化抛物求积
    a = (a + fjifen(0, p) - fjifen(t, p)) * dt / 6
    return a # 返回积分值

```

```

67

69 def A(t): # 为了方便，把电磁四矢量存为一个函数
    return A0 * csin(w * t) * csin(w * t / (4 + 0j)) ** 2 # 返回题目
        中A(t)

71

73 def Sdd(t, p): # S函数的二阶导函数，二阶导的解析表示就是下面方程右边
    a = (p + A(t)) * w * A0 * (ccos(w * t) * csin(w * t / 4) ** 2 + 1
        / 4 * csin(w * t) * csin(w * t / 2))
    return a

75

77
78 def m(t, p): # 返回鞍点近似下每一个M求和中的一项
79     a = exp(S(t, p)) / Sdd(t, p)
80     return a

81

83 def mdi(t, p): # 直接求积分算MDI的被积函数
    q11 = p + A(t)
    a = (-1) * q11 * w * A0 * (ccos(w * t) * csin(w * t / 4) ** 2 + 1
        / 4 * csin(w * t) * csin(w * t / 2))
    b = math.pi * (q11 ** 2 + 2 * Ip) ** 3
    d = a * exp(S(t, p)) / b
    return d

85

89

91 def mjifen(t, p): # 复化抛物求积求MDI
    dt = t / 5000 # 分成10000份
    a = 0
    for i in range(1, 5001, 1):
        a += 4 * mdi((i - 0.5) * dt, p) + 2 * mdi(i * dt, p)
    a = (a + mdi(0, p) - mdi(t, p)) * dt / 6
    return a * 2 ** (7 / 2) # 返回MDI积分的值
93
95
97

```

2.3.2 第 (1) 问

第一问需要通过对区间划分，求解鞍点方程 6 个解。端点的取法是这样的：把解空间分割成 200×60 份，实部在 0 到 $\frac{4\pi}{\omega}$ 之间分 200 份，然后虚部在 0 到 150 之间分 60 份，依次寻找解。每个区间内循环次数控制在 100 以内，根据经验，这样一般能收敛。需要注意的是，为了防止各种各样的运行问题（因为还要包括溢出），这里需要很多条件判断语句来判断解是否有效，如果执行的话，赋 $k=1$ ，在最后加入解向量的时候，如果 $k=1$ ，就舍弃；如果 $k=0$ ，就认为是有效解。具体实现如下：

```

1 nj = 0 # 解的个数

```

```

1   for i in range(0, Nx, 1):
2       for j in range(0, Ny, 1): # 在解空间切割小空间, 产生端点
3           a = complex(i * tf / Nx, j * ti / Ny) # 初始两个端点随机数产
4               生
5           b = complex((i + 1) * tf / Nx, (j + 1) * ti / Ny)
6           x0 = complex(random.uniform(i * tf / Nx, (i + 1) * tf / Nx),
7                           random.uniform(j * ti / Ny, (j + 1) * ti / Ny))
8           sd = 0 # 存放while循环步数
9           k = 0 # 存放判定数k, 在while循环里, 一旦出现判定结果为break的
10              条件, k=1
11           while sd == 0 or math.sqrt(f(x0).real ** 2 + f(x0).imag ** 2)
12               > 10 ** (-12):
13               x0 = x0 - (b - a) * f(x0) / (f(b) - f(a))
14               sd += 1
15           try:
16               ans = f(x0)
17           except OverflowError: # 防止溢出, 出现溢出, break, k=1
18               k = 1
19               break
20           try:
21               ans = math.sqrt(f(x0).real ** 2 + f(x0).imag ** 2)
22           except OverflowError:
23               k = 1
24               break
25           if sd > 100 or abs(x0.imag) > 100 * math.pi: # 循环次数控
26               制在100以内, 一般能收敛, 100次就足够用
27               k = 1
28               break
29           if x0.imag > 0 and x0.real > 0 and x0.real < 4 * math.pi / w
30               and k == 0 and jiansuo(x0, nj, jie) == 0: # 满足t的各个条
31               件才能把x赋在解向量里
32               print(sd) # 输出步数, 一般可以证明100步以内即可
33               jie[nj] = x0 # 存入解向量之中
34               nj += 1
35               print(x0)
36
37 print(jie)

```

2.3.3 第（2）问 & 第（3）问

这一部分主要是利用第（1）问中同样的方法，利用上一个 p 值下求到的解空间来高效的寻找下一个 p 值下的解，即在上一个 p 值下的解的附近寻找新的解。此处仍然是利用弦切法，寻找解的迭代次数经过测试在 20 次以内即可达到较高收敛精度。至于各种数理方程积分、求和的过程中（其中主要是积分的是 $S(t)$ 和 $M_p^0(t_f, t_i)_{DI}$ 函数）， $M_p^0(t_f, t_i)_{DI}$ 积分区间选择分成 5000 份， $S(t)$ 积分区间至少分成 1000 份，当然这也一

定程度上会影响到算法速度（此处就不考虑了）。另外，需要指出，复平面上积分与路径无关的条件是没有奇点，显然这些函数在复平面上都有定义且除了无穷之外没有奇点，可以进行积分。对最后的积分、求和结果进行绘图即可，具体的代码实现如下：

```

1  for l in range(1, 200, 1):
2      nj = 0
3      while nj < 6:
4          for i in range(6):
5              if jie[l - 1][i].real < 1:
6                  a = jie[l - 1][i] # 直接取上一步的值
7              else:
8                  a = jie[l - 1][i] - 1 - 1j
9                  b = jie[l - 1][i] + 1 + 1j # 另一个端点取稍大的值
10                 x0 = complex(random.uniform(a.real, b.real), random.
11                             uniform(a.imag, b.imag))
12                 # x0=jie[l-1][i]
13                 sd = 0
14                 k = 0
15                 while sd == 0 or math.sqrt(f(x0, 1).real ** 2 + f(x0, 1).
16                     imag ** 2) > 10 ** (-12):
17                     x0 = x0 - (b - a) * f(x0, 1) / (f(b, 1) - f(a, 1))
18                     sd += 1
19                     try:
20                         ans = f(x0, 1)
21                     except OverflowError: # 防止溢出，出现溢出，break，k
22                         =1
23                         k = 1
24                         break
25                     try:
26                         ans = math.sqrt(f(x0, 1).real ** 2 + f(x0, 1).imag
27                             ** 2)
28                     except OverflowError:
29                         k = 1
30                         break
31                         if sd > 20 or abs(x0.imag) > 100 * math.pi: # 循环次
32                             数控制在20以内，一般能收敛，20次就足够用
33                             k = 1
34                             break
35
36                 if x0.imag > 0 and 0 < x0.real < 4 * math.pi / w and k ==
37                     0 and jiansuo(x0, 1, nj) == 0: # 满足t的各个条件才能把
38                     x赋在解向量里
39                     print(sd) # 输出步数，一般可以证明20步以内即可
40                     jie[l][nj] = x0 # 存入解向量之中
41                     nj += 1
42                     print(x0)
```

```

37 for i in range(0, 200, 1):
38     a = 0.0
39     print(i)
40     for j in range(0, 6, 1):
41         a += m(jie[i][j], p[i]) # 每个pz, 6个解
42     MSPM[i] = (a.real ** 2 + a.imag ** 2) / 2 # 得到的MSPM模方, 前面
43     系数根号2在这里统一除掉
44     b = mjifen(4 * math.pi / w, p[i]) # 得到MDI模方
45     MDI[i] = b.real ** 2 + b.imag ** 2
46
47 plt.plot(Ek, MSPM, label="MSPM")
48 plt.plot(Ek, MDI, label="MDI")
49 plt.xlabel('Ek')
50 plt.ylabel('|M|^2')
51 plt.legend()
52 plt.show()

```

2.4 结果展示

2.4.1 (1)

(73.87985704438765+108.16876190564986j)
 (24.059818099417274+119.54353926967414j)
 (416.7319373058881+27.87979811303743j)
 (263.983373857842+36.80784948549102j)
 (677.1009140604638+64.20228674143817j)
 (750.6437460397624+84.50511547795732j)

2.4.2 (2)

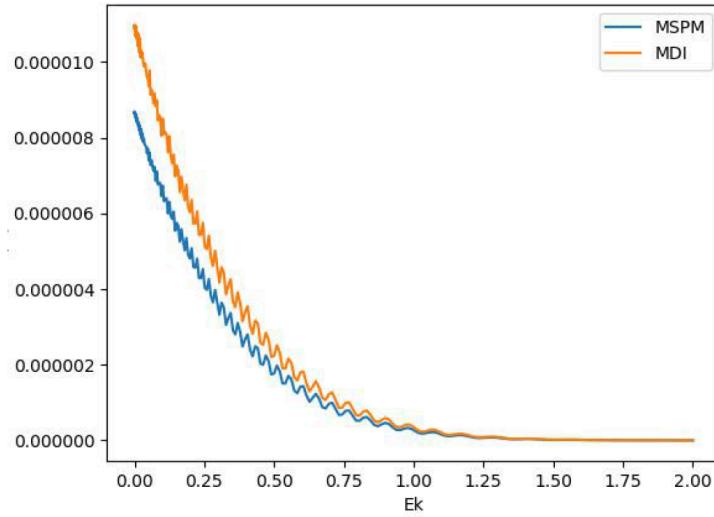


图 2.1: 等间隔原参数做图

可见，除了一个常数的差距外其他细节均符合预期。此外，由于等分间隔绘图效果不好且时间成本较高，所以我还尝试了对做图方法进行调整，即没有完全按照老师所给的 p 值等间隔 0.01 做图。我将 p 值 0-2 的区间划分成 3 份：0-0.2, 0.2-1 和 1-2，横坐标能量的间隔分别设为 0.001, 0.003 和 0.015。最后还对一些题目中的所给参数进行细微调整从而较好的展示“振荡”规律，具体代码可参考“HW2-2（变间隔）.py”，此处仅仅把结果展示出来提供参考：

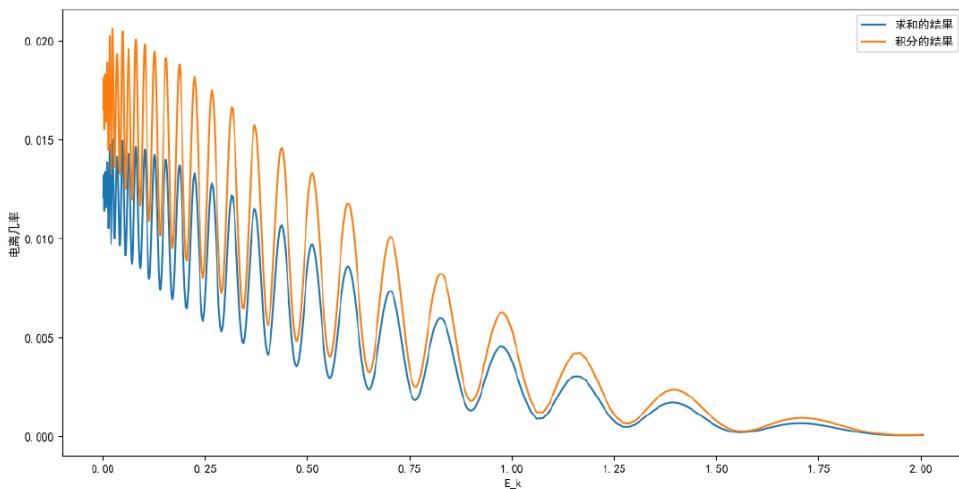


图 2.2: 变间隔调整参数做图

Chapter 3

第三题

3.1 题目

随机过程是由 Pearson 于 1906 年提出的，是蒙特卡洛方法的重要应用，考虑醉汉从原点出发，向左为负，向右为正，步长为 1，每一步都和前一步无关，向左走一步的概率是 q ，向右走一步的概率是 p 。

3.2 问题 (1)

表示醉汉走了 3 步之后，离原点的距离 x 的概率以及 x 的期望值

3.2.1 解

解：走三步之后的距离 x 共有 4 种可能，因为为奇数步，所以只能到奇数位置将位置和概率列表表示在下面：

x	-3	-1	1	3
P	q^3	$3q^2p$	$3p^2q$	p^3

可以计算期望： $E(x) = 3(p^3 - q^3) + 3p^2q - 3q^2p$

3.3 问题 (2)

在二维平面上考虑上述问题，醉汉从原点出发，且向上向下向左向右的概率是相同的。编写程序描述这一过程，输出游走五十步，每一步的坐标位置以及距离远点的位移大小 R 。

3.3.1 思路

为了保存位置，设置两个变量 x 和 y 即可，然后用随机数生成函数生成 1-4 的整数，生成 50 个，对每个数字指定一个方向，如果向右则 x 加 1，向上 y 加 1，向左向下反之，这样每循环依次输出一个位置以及距离 $R = \sqrt{x^2 + y^2}$ 即可。

3.3.2 程序实现

按照之前的思路，首先初始化x和y的值，然后设置for循环得到一个1-4间的整数，根据这个数确定行走方向，在本程序中，1代表向左，2代表向右，3代表向上，4代表向下，求距离 $R = \sqrt{x^2 + y^2}$ 后随即输出即可。

```
1 # 本程序用来解决第三题第二问
2 import math
3 import random
4
5 x = 0
6 y = 0 # 初始在原点
7 for i in range(1, 51):
8     a = random.randint(1, 4)
9     if a==1: # 1代表向左
10        x = x - 1
11    elif a==2: # 2代表向右
12        x = x + 1
13    elif a==3: # 3代表向上
14        y = y + 1
15    elif a==4: # 4代表向下
16        y = y - 1
17 R = math.sqrt(x*x+y*y) # 求距离
18 print('第', i, '步行走后：', 'x坐标为：', x, 'y坐标为：', y, '与原'
19      点距离：', R, '\n')
```

3.3.3 结果

第 1 步行走后: x 坐标为: 0 y 坐标为: -1 与原点距离: 1.0
第 2 步行走后: x 坐标为: 0 y 坐标为: -2 与原点距离: 2.0
第 3 步行走后: x 坐标为: 1 y 坐标为: -2 与原点距离: 2.23606797749979
第 4 步行走后: x 坐标为: 1 y 坐标为: -3 与原点距离: 3.1622776601683795
第 5 步行走后: x 坐标为: 2 y 坐标为: -3 与原点距离: 3.605551275463989
第 6 步行走后: x 坐标为: 2 y 坐标为: -4 与原点距离: 4.47213595499958
第 7 步行走后: x 坐标为: 1 y 坐标为: -4 与原点距离: 4.123105625617661
第 8 步行走后: x 坐标为: 0 y 坐标为: -4 与原点距离: 4.0
第 9 步行走后: x 坐标为: 0 y 坐标为: -5 与原点距离: 5.0
第 10 步行走后: x 坐标为: 0 y 坐标为: -4 与原点距离: 4.0
第 11 步行走后: x 坐标为: 0 y 坐标为: -5 与原点距离: 5.0
第 12 步行走后: x 坐标为: 1 y 坐标为: -5 与原点距离: 5.0990195135927845
第 13 步行走后: x 坐标为: 1 y 坐标为: -6 与原点距离: 6.082762530298219
第 14 步行走后: x 坐标为: 2 y 坐标为: -6 与原点距离: 6.324555320336759
第 15 步行走后: x 坐标为: 2 y 坐标为: -5 与原点距离: 5.385164807134504

第 16 步行走后: x 坐标为: 3 y 坐标为: -5 与原点距离: 5.830951894845301
第 17 步行走后: x 坐标为: 3 y 坐标为: -4 与原点距离: 5.0
第 18 步行走后: x 坐标为: 2 y 坐标为: -4 与原点距离: 4.47213595499958
第 19 步行走后: x 坐标为: 2 y 坐标为: -5 与原点距离: 5.385164807134504
第 20 步行走后: x 坐标为: 3 y 坐标为: -5 与原点距离: 5.830951894845301
第 21 步行走后: x 坐标为: 2 y 坐标为: -5 与原点距离: 5.385164807134504
第 22 步行走后: x 坐标为: 3 y 坐标为: -5 与原点距离: 5.830951894845301
第 23 步行走后: x 坐标为: 4 y 坐标为: -5 与原点距离: 6.4031242374328485
第 24 步行走后: x 坐标为: 4 y 坐标为: -6 与原点距离: 7.211102550927978
第 25 步行走后: x 坐标为: 4 y 坐标为: -5 与原点距离: 6.4031242374328485
第 26 步行走后: x 坐标为: 5 y 坐标为: -5 与原点距离: 7.0710678118654755
第 27 步行走后: x 坐标为: 5 y 坐标为: -6 与原点距离: 7.810249675906654
第 28 步行走后: x 坐标为: 5 y 坐标为: -5 与原点距离: 7.0710678118654755
第 29 步行走后: x 坐标为: 6 y 坐标为: -5 与原点距离: 7.810249675906654
第 30 步行走后: x 坐标为: 7 y 坐标为: -5 与原点距离: 8.602325267042627
第 31 步行走后: x 坐标为: 7 y 坐标为: -6 与原点距离: 9.219544457292887
第 32 步行走后: x 坐标为: 8 y 坐标为: -6 与原点距离: 10.0
第 33 步行走后: x 坐标为: 9 y 坐标为: -6 与原点距离: 10.816653826391969
第 34 步行走后: x 坐标为: 10 y 坐标为: -6 与原点距离: 11.661903789690601
第 35 步行走后: x 坐标为: 9 y 坐标为: -6 与原点距离: 10.816653826391969
第 36 步行走后: x 坐标为: 8 y 坐标为: -6 与原点距离: 10.0
第 37 步行走后: x 坐标为: 8 y 坐标为: -5 与原点距离: 9.433981132056603
第 38 步行走后: x 坐标为: 8 y 坐标为: -4 与原点距离: 8.94427190999916
第 39 步行走后: x 坐标为: 9 y 坐标为: -4 与原点距离: 9.848857801796104
第 40 步行走后: x 坐标为: 9 y 坐标为: -3 与原点距离: 9.486832980505138
第 41 步行走后: x 坐标为: 10 y 坐标为: -3 与原点距离: 10.44030650891055
第 42 步行走后: x 坐标为: 11 y 坐标为: -3 与原点距离: 11.40175425099138
第 43 步行走后: x 坐标为: 12 y 坐标为: -3 与原点距离: 12.36931687685298
第 44 步行走后: x 坐标为: 13 y 坐标为: -3 与原点距离: 13.341664064126334
第 45 步行走后: x 坐标为: 14 y 坐标为: -3 与原点距离: 14.317821063276353
第 46 步行走后: x 坐标为: 14 y 坐标为: -2 与原点距离: 14.142135623730951
第 47 步行走后: x 坐标为: 14 y 坐标为: -1 与原点距离: 14.035668847618199
第 48 步行走后: x 坐标为: 15 y 坐标为: -1 与原点距离: 15.033296378372908
第 49 步行走后: x 坐标为: 15 y 坐标为: 0 与原点距离: 15.0
第 50 步行走后: x 坐标为: 14 y 坐标为: 0 与原点距离: 14.0

3.4 问题 (3)

考虑醉汉从原点出发，可以向任意方向行走，步长依旧是1，每一步的位移方向与x轴正方向的夹角为 θ ,那么n步之后x, y方向的位移以及距离原点的距离为:

$$X_n = \sum_{k=1}^n \cos \theta_k \quad Y_n = \sum_{k=1}^n \sin \theta_k \quad R_n = \sqrt{X_n^2 + Y_n^2}$$

编写程序描述上述过程，输出游走100步，每一步距离原点的位移大小R以及R的期望值，做出期望值与N关系的散点图，用函数 $E(R) = a * \sqrt{N}$ 来拟合散点图，输出参数a。

3.4.1 思路

求每一步距离原点距离R与上一题类似，只不过是通过计算随机数 θ 来计算每一步的位置，最后生成一个列表储存每一步之后的距离R，期望值通过多次运行上述过程求平均得到，再生成一个 \sqrt{N} 的列表，N从1到100，最后根据最小二乘法程序回归R与 \sqrt{N} 得到参数a。

3.4.2 程序实现

首先是函数的定义，程序中定义了两个函数，walk函数用来进行一次随机行走，基本原理与（2）中相同，只不过是通过生成角度来预测坐标的，每完成随机行走后，将100次行走的每次的距离R做成一个列表返回。

linefit是最小二乘法线性回归函数，输入两列数据，进行回归然后输出a, b, r，本程序中得到a就够了。

```
1 import math
3 import random
5 import matplotlib.pyplot as plt
7
7 def walk(x, y):
9     r = []
11    for i in range(1, 101):
13        theta = random.uniform(0, 2 * math.pi) # 随机输出角度
15        x = x + math.cos(theta)
17        y = y + math.sin(theta)
19        R = math.sqrt(x * x + y * y) # 求距离
21        r.append(R) # r是储存距离的list
23        print('第', i, '步行走后：', 'x坐标为：', x, 'y坐标为：', y, ','
25        '与原点距离：', R, '\n')
27
29
31 def linefit(x, y): # 最小二乘法函数
33     N = float(len(x))
35     sx, sy, sxx, syy, sxy = 0, 0, 0, 0, 0
37     for i in range(0, int(N)):
39         sx += x[i]
41         sy += y[i]
43         sxx += x[i] * x[i]
45         syy += y[i] * y[i]
```

```

27         sxy += x[i]*y[i]
28         a = (sy*sx/N-sxy)/( sx*sx/N -sxx )
29         b = (sy - a*sx)/N
30         r = abs(sy*sx/N-sxy)/math.sqrt((sxx-sx*sx/N)*(syy-sy*sy/N))
31         return a,b,r

33
34         x = 0
35         y = 0 # 初始在原点
36         r = [0] * 100
37         step = 1000
38         for j in range(step):
39             a = walk(x, y)
40             r = [r[i] + a[i] for i in range(min(len(r), len(a)))]
41         r = [r[i] / step for i in range(len(r))]
42         n = [math.sqrt(i) for i in range(1, 101)]
43         N = [i for i in range(1, 101)]
44         print(r)
45         plt.scatter(N, r)
46         plt.xlabel('N')
47         plt.ylabel('E(r)')
48         plt.show()
49         aa = linefit(n, r)
50         print('回归系数为: ',aa[0])

```

在主程序里，也是首先设置原点，然后多次运行walk程序每次输出一个r向量，然后将他们求平均，程序里设置了一个求平均次数的变量step，在这里设置为5000，求平均后输出，然后生成另外一个 \sqrt{N} 的向量，用两个向量做回归，得到回归系数a输出。

3.4.3 结果展示

得到的回归系数是: 0.8898790984845919

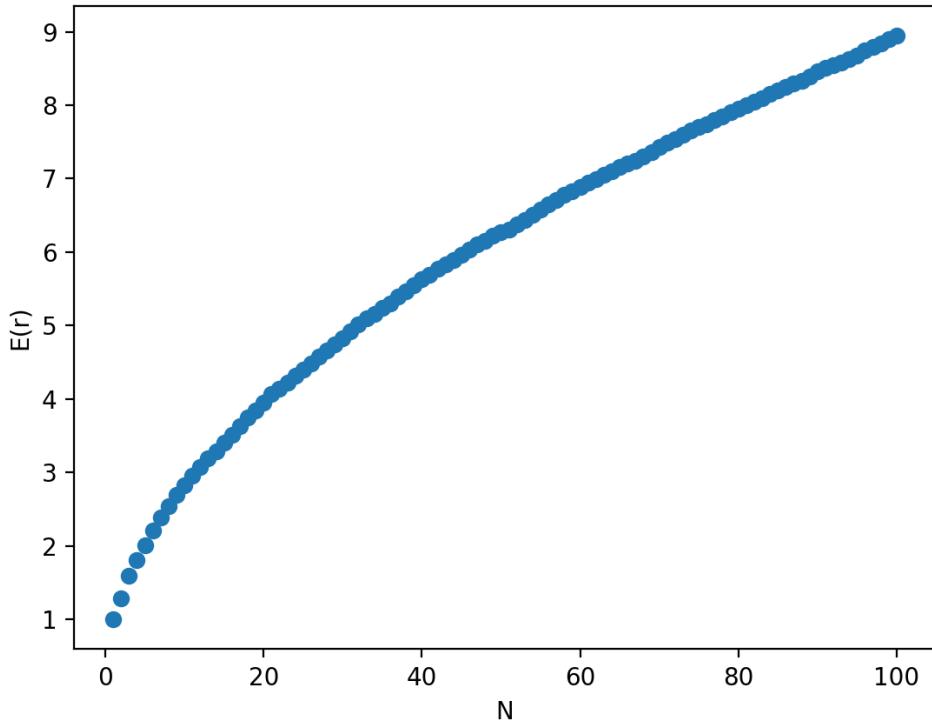


图 3.1: 随机行走距离期望与 N 的关系

第 1 步行走后: x 坐标为: -0.9316492981750749 y 坐标为: -0.36335875551566166 与原点距离: 1.0

第 2 步行走后: x 坐标为: -1.0508668335032652 y 坐标为: 0.6295094025346767 与原点距离: 1.2249910977785696

第 3 步行走后: x 坐标为: -2.017179359082377 y 坐标为: 0.3721378755373844 与原点距离: 2.0512189461677335

第 4 步行走后: x 坐标为: -2.689720553724583 y 坐标为: 1.112197563339142 与原点距离: 2.910597889957663

第 5 步行走后: x 坐标为: -3.655713551565143 y 坐标为: 0.8536293361282026 与原点距离: 3.754054423499439

第 6 步行走后: x 坐标为: -4.651982795853852 y 坐标为: 0.9399287734635811 与原点距离: 4.745988836070412

第 7 步行走后: x 坐标为: -4.810767800865883 y 坐标为: 1.9272419568979609 与原点距离: 5.182446178618341

第 8 步行走后: x 坐标为: -5.275091223214426 y 坐标为: 2.8129076723112516 与原点距离: 5.978213527984876

第 9 步行走后: x 坐标为: -4.514906579362201 y 坐标为: 2.1632005788827864 与原点距离: 5.0063777489165675

第 10 步行走后: x 坐标为: -3.52014232514429 y 坐标为: 2.265396853424883 与原点距离: 4.186099006566807

第 11 步行走后: x 坐标为: -4.248475995696487 y 坐标为: 1.5801742149051365 与原点距离: 4.532824597914674

第 12 步行走后: x 坐标为: -3.2560199777880374 y 坐标为: 1.702775407873651 与原点距离: 3.674385688168036

第 13 步行走后: x 坐标为: -2.9349352194833176 y 坐标为: 2.649825870090105 与原点距离: 3.954165131144926

第 14 步行走后: x 坐标为: -3.4723939909303505 y 坐标为: 1.8065358434713588 与原点距离: 3.9142166243063223

第 15 步行走后: x 坐标为: -2.649647838902371 y 坐标为: 1.2381268627865418 与原点距离: 2.9246524235460267

第 16 步行走后: x 坐标为: -1.99949202915133 y 坐标为: 1.9979277729248281 与原点距离: 2.82660286571426

第 17 步行走后: x 坐标为: -1.29806633715891 y 坐标为: 1.2851851836171915 与原点距离: 1.8266573767004866

第 18 步行走后: x 坐标为: -0.31716227846296685 y 坐标为: 1.0906929533512355 与原点距离: 1.1358710443399203

第 19 步行走后: x 坐标为: -1.0037023230609106 y 坐标为: 0.3636009597469506 与原点距离: 1.067531737817088

第 20 步行走后: x 坐标为: -1.9992213987928427 y 坐标为: 0.4581618917106285 与原点距离: 2.051048151654922

第 21 步行走后: x 坐标为: -1.173420408867298 y 坐标为: 1.0221235260777104 与原点距离: 1.5561657876035666

第 22 步行走后: x 坐标为: -2.169274997570837 y 坐标为: 1.113083067402986 与原点距离: 2.438177173633039

第 23 步行走后: x 坐标为: -2.1751084636751803 y 坐标为: 2.1130660525948386 与原点距离: 3.0325146283868336

第 24 步行走后: x 坐标为: -1.2112741267992464 y 坐标为: 2.379568155886593 与原点距离: 2.6701179035321263

第 25 步行走后: x 坐标为: -1.1220408120819538 y 坐标为: 1.38355740516923 与原点距离: 1.7813496785797402

第 26 步行走后: x 坐标为: -1.9949779056805208 y 坐标为: 1.8713901947802674 与原点距离: 2.7353314434037355

第 27 步行走后: x 坐标为: -1.5420127287760013 y 坐标为: 0.9798619851806736 与原点距离: 1.8270010305715265

第 28 步行走后: x 坐标为: -2.5363811227795905 y 坐标为: 0.8738832278435422 与原点距离: 2.6827040641671425

第 29 步行走后: x 坐标为: -1.6738178089911762 y 坐标为: 1.3798323653132583 与原点距离: 2.1692402849988754

第 30 步行走后: x 坐标为: -0.7033041419590255 y 坐标为: 1.620878464854005 与原点距离: 1.7668853143099574

第 31 步行走后: x 坐标为: -1.663373798966801 y 坐标为: 1.3411173994911274 与原点距离: 2.1366816033997886

第 32 步行走后: x 坐标为: -2.2769466903296576 y 坐标为: 2.130755485568936 与原点距离: 3.11842992063078

第 33 步行走后: x 坐标为: -2.4794254789647865 y 坐标为: 3.110042134182533 与原点距离: 3.9774253961740635

第 34 步行走后: x 坐标为: -3.4638444104777077 y 坐标为: 2.9342030961990444 与原点距离: 4.539577723725161

第 35 步行走后: x 坐标为: -2.5150265465355233 y 坐标为: 2.618379314535614 与原点距离: 3.630601708335189

第 36 步行走后: x 坐标为: -1.8840551236426804 y 坐标为: 3.3941853879151545 与原点距离: 3.8820301591385125

第 37 步行走后: x 坐标为: -2.5795617841078315 y 坐标为: 4.112705036392921 与原点距离: 4.8547380685677854

第 38 步行走后: x 坐标为: -2.1427372427372995 y 坐标为: 5.012251766727941 与原点距离: 5.451054087466094

第 39 步行走后: x 坐标为: -1.531601275381226 y 坐标为: 5.803777400806684 与原点距离: 6.00246904072514

第 40 步行走后: x 坐标为: -2.437244956881362 y 坐标为: 5.379737871801568 与原点距离: 5.906076747650652

第 41 步行走后: x 坐标为: -3.4371306873326004 y 坐标为: 5.364620794041849 与原点距离: 6.371265465013185

第 42 步行走后: x 坐标为: -4.288708202335536 y 坐标为: 5.888849499800534 与原点距离: 7.285023436893055

第 43 步行走后: x 坐标为: -3.2894963746976362 y 坐标为: 5.92854488377057 与原点距离: 6.780002274190703

第 44 步行走后: x 坐标为: -3.4112746912743854 y 坐标为: 6.921102207864112 与原点距离: 7.716116302326621

第 45 步行走后: x 坐标为: -3.6350022981131747 y 坐标为: 7.895753918957972 与原点距离: 8.692305313093197

第 46 步行走后: x 坐标为: -4.418873050912853 y 坐标为: 7.274829895501587 与原点距离: 8.51173243520774

第 47 步行走后: x 坐标为: -4.907988895708826 y 坐标为: 8.147048727200103 与原点距离: 9.511191195837352

第 48 步行走后: x 坐标为: -4.538791559289904 y 坐标为: 7.217697702677521 与原点距离: 8.5261825541046

第 49 步行走后: x 坐标为: -4.477541264539326 y 坐标为: 8.215820140754618 与原点距离: 9.356712903626018

第 50 步行走后: x 坐标为: -3.5138909414022224 y 坐标为: 8.482986857839204 与原点距离: 9.181965779632446

第 51 步行走后: x 坐标为: -2.5269276788929638 y 坐标为: 8.643932558176557 与原点距离: 9.00571671576783

第 52 步行走后: x 坐标为: -1.5673880246909802 y 坐标为: 8.925506086786648 与原点距离: 9.06208387321648

第 53 步行走后: x 坐标为: -2.562106821155841 y 坐标为: 9.028143876900709 与原点距离: 9.384656265683525

第 54 步行走后: x 坐标为: -2.4995134140593738 y 坐标为: 8.030104766751183 与原点距离: 8.410121870226543

第 55 步行走后: x 坐标为: -3.4967111848881105 y 坐标为: 8.10491523295878 与原点距离: 8.827040276557533

第 56 步行走后: x 坐标为: -2.825291143196375 y 坐标为: 7.363838227773307 与原点距离: 7.887229138844605

第 57 步行走后: x 坐标为: -1.855351520499606 y 坐标为: 7.607183924926245 与原点距离: 7.830170913350262

第 58 步行走后: x 坐标为: -0.8554391954788008 y 坐标为: 7.593942238044001 与原点距离: 7.641971926925023

第 59 步行走后: x 坐标为: -1.6253005753028409 y 坐标为: 6.955731100680021 与原点距离: 7.143094364842666

第 60 步行走后: x 坐标为: -2.017727803380783 y 坐标为: 7.8755141569835505 与原点距离: 8.12987999452539

第 61 步行走后: x 坐标为: -2.308863954181419 y 坐标为: 6.918832526713067 与原点距离: 7.293908163092059

第 62 步行走后: x 坐标为: -3.034893763279382 y 坐标为: 6.231169297871235 与原点距离: 6.930948778711699

第 63 步行走后: x 坐标为: -2.7957239415977178 y 坐标为: 5.260191545089658 与原点距离: 5.956986440194018

第 64 步行走后: x 坐标为: -3.6929924377980563 y 坐标为: 5.701676817467325 与原点距离: 6.793181263331485

第 65 步行走后: x 坐标为: -4.655374319412896 y 坐标为: 5.973377229574998 与原点距离: 7.573225573073482

第 66 步行走后: x 坐标为: -3.8244236309137096 y 坐标为: 5.417031156269248 与原点距离: 6.631021237839835

第 67 步行走后: x 坐标为: -3.729232503689878 y 坐标为: 4.421572141895549 与原点距离: 5.78424370791583

第 68 步行走后: x 坐标为: -3.2847756712995384 y 坐标为: 5.3173724121241 与原点距离: 6.250136044917727

第 69 步行走后: x 坐标为: -2.667721572556422 y 坐标为: 6.104293144615195 与原点距离: 6.661766521282398

第 70 步行走后: x 坐标为: -3.574802062612742 y 坐标为: 6.52525036875939 与原点距离: 7.440302558488792

第 71 步行走后: x 坐标为: -4.464964762270385 y 坐标为: 6.980893176382003 与原点距离: 8.286662770281717

第 72 步行走后: x 坐标为: -5.412842212376541 y 坐标为: 7.299528296198152 与原点距离: 9.087462471068745

第 73 步行走后: x 坐标为: -5.585917586919274 y 坐标为: 6.314619714284636 与原点距离: 8.43071156094705

第 74 步行走后: x 坐标为: -4.644278772335106 y 坐标为: 5.9779947852188835 与原点距离: 7.570055942149083

第 75 步行走后: x 坐标为: -5.583902545710192 y 坐标为: 6.320204026047307 与原点距离: 8.43356072906654

第 76 步行走后: x 坐标为: -5.098409804607875 y 坐标为: 5.445963327877199 与原点距离: 7.460046856709749

第 77 步行走后: x 坐标为: -4.329851630254366 y 坐标为: 6.085743237404218 与原点距离: 7.468861083969065

第 78 步行走后: x 坐标为: -3.747213694560105 y 坐标为: 6.898475072568764 与原点距离: 7.850513919454662

第 79 步行走后: x 坐标为: -3.2735835939283637 y 坐标为: 7.779198938360742 与原点距离: 8.4399221364317

第 80 步行走后: x 坐标为: -2.3043440794008894 y 坐标为: 8.025318347344564 与原点距离: 8.349594972960977

第 81 步行走后: x 坐标为: -1.3996778342316052 y 坐标为: 8.451439204439483 与原点距离: 8.566558390974519

第 82 步行走后: x 坐标为: -1.8061583066537965 y 坐标为: 9.365098673574145 与原点距离: 9.537676918121873

第 83 步行走后: x 坐标为: -2.6505805153262894 y 坐标为: 9.900776873149852 与原点距离: 10.249437046010202

第 84 步行走后: x 坐标为: -3.4454321701806196 y 坐标为: 10.507580669096559 与原点距离: 11.058040258422249

第 85 步行走后: x 坐标为: -4.332349459804256 y 坐标为: 10.045652404788493 与原点距离: 10.940035835393733

第 86 步行走后: x 坐标为: -4.309608218021032 y 坐标为: 9.045911020268397 与原点距离: 10.020041376134513

第 87 步行走后: x 坐标为: -3.3665488677515087 y 坐标为: 8.71328651445257 与原点距离: 9.341039190684299

第 88 步行走后: x 坐标为: -2.6097595851354694 y 坐标为: 8.059627595965642 与原点距离: 8.471625704542037

第 89 步行走后: x 坐标为: -1.723984529600382 y 坐标为: 8.52374239964229 与原点距离: 8.696338721195325

第 90 步行走后: x 坐标为: -2.7221480068172212 y 坐标为: 8.584320223558207 与原点距离: 9.005589568240904

第 91 步行走后: x 坐标为: -2.0420223903072565 y 坐标为: 9.317415814710508 与原点距离: 9.538558219496993

第 92 步行走后: x 坐标为: -1.0426084649450167 y 坐标为: 9.283184165728327 与原点距离: 9.34154915771491

第 93 步行走后: x 坐标为: -2.016098127183826 y 坐标为: 9.511915181364968 与原点距离: 9.723228994316418

第 94 步行走后: x 坐标为: -2.2809076011994285 y 坐标为: 10.476215934486929 与原点距离: 10.721643520990947

第 95 步行走后: x 坐标为: -1.7333445738569666 y 坐标为: 11.312980376302878 与原点距离: 11.44499927506915

第 96 步行走后: x 坐标为: -1.8111847520603512 y 坐标为: 12.3099462265993 与原点距离: 12.442474283994413

第 97 步行走后: x 坐标为: -0.9661936754999443 y 坐标为: 11.775165822570427 与原点距离: 11.814739115511895

第 98 步行走后: x 坐标为: 0.0337948076204746 y 坐标为: 11.770366486520123 与原点距离: 11.770415001860307

第 99 步行走后: x 坐标为: 0.930947728306026 y 坐标为: 12.212086575383404 与原点距离: 12.247519022132511

第 100 步行走后: x 坐标为: 0.36470286118035666 y 坐标为: 13.036323647137541 与原点距离: 13.041424094395158

3.5 问题 (4)

解析证明随机变量 X_n 和 Y_n 的期望值为 0, 计算他们的方差。

3.5.1 解

因为向任何方向走的概率是相同的，所以在 $\theta - \theta + d\theta$ 方向的概率是 $d\theta/2\pi$ ，那么对于 X_n 的期望可以化成积分形式：

$$E(X_n) = N * \int_0^{2\pi} \cos\theta * \frac{d\theta}{2\pi} = 0$$

同理对 Y_n 也有：

$$E(Y_n) = N * \int_0^{2\pi} \sin\theta * \frac{d\theta}{2\pi} = 0$$

求方差：

$$(\bar{X}^2 - \bar{X}^2) = N * \bar{X}^2 = \int_0^{2\pi} \cos^2\theta * \frac{d\theta}{2\pi} = N * \frac{1}{2}$$

同理对于Y也有：

$$(\bar{Y}^2 - \bar{Y}^2) = N * \bar{Y}^2 = \int_0^{2\pi} \sin^2\theta * \frac{d\theta}{2\pi} = N * \frac{1}{2}$$

3.6 问题 (5)

说明 X, Y 是否是相关的，他们是否是独立的。

3.6.1 解

当 n 为有限步时，X 和 Y 不是相关的，他们不是独立的，因为给定一个 X，那么 Y 也有一个可能的范围，不会大于 $\sqrt{n^2 - X^2}$ 。但是当 n 趋于正无穷时，X 和 Y 趋近于互相独立。

3.7 问题 (6)

试解释为什么 n 足够大时， (X_n, Y_n) 的联合概率密度可以表示成两个独立的正态分布变量的乘积的形式。

3.7.1 解

因为 X, Y 方向是独立的，所以 (X_n, Y_n) 的联合概率密度可以表示成他们的乘积的形式，只要解释 n 足够大时，X, Y 的分布都满足正态分布即可。我们知道 X 和 Y 的每一步都是独立的，可以看成随机变量，对于 X 来说，X 可以看成大量 $\cos\theta$ 的相加， θ 是等几率分布的，对于 $\cos\theta$ 这个随机变量，他的期望 $E(\cos\theta) = 0$ ，平方的期望 $E^2(\cos\theta) = \frac{1}{2}$ ，那么根据中心极限定理，只要一个随机变量有有限的期望和方差，当 $n \rightarrow \infty$ 时就满足正态分布，因对 X 来说，他的期望是 0，方差是 $\frac{N}{2}$ 所以得到的正态分布是：

$$P(n) = \frac{1}{\sqrt{2\pi(\Delta n)^2}} e^{-\frac{(n-\bar{n})^2}{2(\Delta n)^2}}$$

对于 X 可以写为：

$$X_n = \frac{1}{\sqrt{\pi N}} e^{-\frac{x^2}{N}}$$

所以 XY 的联合分布就可以表示成两个正态分布的乘积的形式。

3.8 问题 (7)

推断 R_n 的分布函数以及概率密度函数

3.8.1 解

考虑 R_n 和 X, Y 的关系:

$$\begin{cases} X_n = R_n \cos \theta \\ Y_n = R_n \sin \theta \end{cases} \quad (3.1)$$

根据二维变量函数的概率密度之间的关系, 可以通过 $p(x, y)$ (即 X, Y 变量的联合概率密度, 由 (6) 问求得) 求出 $p(R, \theta)$: 雅克比系数矩阵为

$$J\left(\frac{x, y}{R, \theta}\right) = \frac{\partial(x, y)}{\partial(R, \theta)} = \begin{vmatrix} \frac{\partial x}{\partial R} & \frac{\partial y}{\partial R} \\ \frac{\partial x}{\partial \theta} & \frac{\partial y}{\partial \theta} \end{vmatrix} = R \quad (3.2)$$

因此我们可以得到 R_n 和 θ 的联合分布密度 $p(R, \theta)$:

$$p(R, \theta) = |J\left(\frac{x, y}{R, \theta}\right)| p(x)p(y) = \frac{R}{\pi N} e^{-\frac{R^2}{N}} \quad (3.3)$$

在此我们认为 θ 分布遵从 $(0, 2\pi)$ 均匀分布且与径向方向 R_n 分布独立 (在此就不再证明), 概率密度 $p(\theta) = \frac{1}{2\pi}$, 从而我们可以得到 R_n 的概率分布密度 $p(R_n)$

$$p(R) = \frac{2\pi R}{\pi N} e^{-\frac{R^2}{N}} \quad (3.4)$$

分布函数 $F_R(R_s)$ 为

$$F_R(R_s) = \int_0^{R_s} \frac{2\pi R}{\pi N} e^{-\frac{R^2}{N}} dR = -e^{\frac{R_s^2}{N}} + 1 \quad (3.5)$$

3.9 问题 (8)

计算 R_n 的期望值 $E(R_n)$, 与第三问对比。

3.9.1 解

对于连续分布的随机变量, 我们已知 R_n 的随机变量分布密度为 $p(R) = \frac{2\pi R}{\pi N} e^{-\frac{R^2}{N}}$, 则期望:

$$E(R_n) = \int_0^\infty \frac{2\pi R}{\pi N} e^{-\frac{R^2}{N}} R dR = \frac{\sqrt{\pi N}}{2} \approx 0.89\sqrt{N} \quad (3.6)$$

与 (3) 中实验结果相近, 这是合理的。