# 计算物理 2019 春季 第一次大作业

姚铭星

1700011321

## 目录

# 1 第一题：十进制整数与双精度浮点数的相互转换

## 1.1 问题描述

编写两个程序，实现任意十进制数与其规范化二进制双精度浮点数之间的转换，并举足够多的算例说明你程序的正确性

## 1.2 思路

在 IEEE 754 标准下, double 型机器数的编码长度为 64, 格式为

| $(\mathrm{S})_2$ | $(\mathrm{E})_2$ | | | $(\mathrm{T})_2$ | | |
|---|---|---|---|---|---|---|
| a[0] | a[1] | ... | a[11] | a[12] | ... | a[63] |

(1)

$$d = (-)^{\mathrm{S}} \times \begin{cases} 2^{\mathrm{E}-1023}(1+2^{-52}\mathrm{T}) & \text{if } \mathrm{E} > 0 \\ 2^{-1022}(0+2^{-52}\mathrm{T}) & \text{if } \mathrm{E} = 0 \end{cases} \tag{2}$$

对于十进制转二进制数，具体操作如下：

1. 确定符号位
2. 分开计算整数位和小数位
3. 确定指数位
4. 判断是否上溢或者下溢

对于二进制转十进制数，具体操作为：

1. 判断符号，取绝对值
2. 对 $x$ 取 $log_2$，并向下取整，得到指数位 $E$，加上偏移量 1023
3. 计算 $x/2^E - 1$ 得到小数位
4. 将指数位和小数位转化为二进制

## 1.3 实现

```
# 二进制转十进制
def bi2deci(a):
        '''
        实现double型二进制数转十进制
        输入为a[64]数据类型为<class ='list'>
```

```
        输出为float(python默认为double)
        '''
        sign = -1 if a[0]==1 else 1 #符号位
        E = 0# 指数位
        for i in range(1,12):
                E += 2**(11-i)*a[i]
        # print(E)
        T = 1# 尾数位
        for i in range(12,64):
                T += a[i]*2**(11-i)
        # print(T)
        x = sign * T * 2**(E - 1023)
        if x < 2**(-1022):
                return 0
        if x >= 2**(1023):
                return NaN
        return x


# 十进制转二进制

def deci2bi(a):
'''
        实现double(float)型十进制数转二进制
        返回res[64]
        a[0]为符号,a[1:12]为指数位,a[12:64]为小数位
        '''
        S = np.sign(a)
        res = [0 for i in range(64)]
        if S ==0:
                return res
        S = 0 if np.sign(a)==1 else 1
        res[0] = int(S)
        a = abs(a)
        E = ms.floor(ms.log2(a))
        if a >=2:
                # 符号位> 1023
                while a>=2 :
                        a/=2
        elif a<1:
                while a <1:
                        a*=2
```

```
        # 此时 a \in (1,2]
        a -= 1 #去掉首位的1
        # print("E = "+str(E))
        E +=1023 #加上偏移量
        for k in range(11,0,-1): #计算指数位
                res[k] =int(E %2.0)
                E /=2.0
                if E==0:
                        break
        p = 1
        while p<=52:
                a *=2
                h = 1 if a>=1.0 else 0
                res[11+p] = h
                a -=h
                p+= 1
        return res
```

### 1.4 结果讨论

表格中几乎涵盖了所有上溢、下溢、零、无理数的例子，可以说明程序的进制转换是合理的

表 1: 一些十进制实数与机器数编码间相互转换的例子

| x | deci2bi(x) | bi2deci(deci2bi(x)) |
|---|---|---|
| 0. | 0 00000000000 0000000000000000000000000000000000000000000000000000 | 0 |
| $\pi$ | 0 10000000000 1001001000011111101101010001000100001011010001100 0 | 3.14159$\cdots$ |
| -0.0625 | 1 01111111011 0000000000000000000000000000000000000000000000000000 | -0.0625 |
| $2^{-1022}$ | 0 00000000001 0000000000000000000000000000000000000000000000000000 | 2.22507$\cdots$e-308 |
| $1.5 \times 2^{1023}$ | 0 11111111110 1000000000000000000000000000000000000000000000000000 | 1.34827$\cdots$e+308 |
| $0.5 \times 2^{-1022}$ | 0 00000000000 1000000000000000000000000000000000000000000000000000 | 1.11254$\cdots$e-308 |
| $-2^{1024}$ | 1 11111111111 0000000000000000000000000000000000000000000000000000 | NaN |

## 2 第二题：高斯求积

### 2.1 问题描述

构造以下几种非标准权函数的 Gauss 求积公式的求积系数与节点：

$$\int_0^1 \sqrt{x} f(x)\mathrm{d}x \approx A_0 f(x_0) + A_1 f(x_1) \tag{3}$$

$$\int_{-1}^1 \left(1+x^2\right) f(x)\mathrm{d}x \approx A_0 f(x_0) + A_1 f(x_1) \tag{4}$$

$$\int_0^1 \sqrt{x} f(x)\mathrm{d}x \approx A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2) \tag{5}$$

## 2.2 思路

具有如下形式的积分问题可以用高斯求积来解决：

$$\int_a^b w(x)f(x)dx$$

解决高斯求积问题的关键是找到一组在该权函数下的正交基。在定义内积

$$(f,g) = \int_a^b w(x)f(x)g(x)dx$$

的前提下，权重以及正交基满足下列关系:

$$\begin{aligned} p_{-1}(x) &= 0 \\ p_0(x) &= 1 \\ p_{j+1}(x) &= (x-a_j)p_j(x) - b_j p_{j-1}(x) \end{aligned}$$

其中：

$$\begin{aligned} a_j &= \frac{(xp_j, p_j)}{(p_j, p_j)} \\ b_j &= \frac{(p_j, p_j)}{(p_{j-1}, p_{j-1})} \end{aligned}$$

设 $x_\mu$ 是正交多项式 $p_N(x)$ 的第 $\mu$ 个根，则系数满足：

$$A_j = \frac{(p_{N-1}(x), p_{N-1}(x))}{p_{N-1}(x_j)p'_N(x_j)} \tag{6}$$

## 2.3 伪代码

求解流程大概按下面进行

1. 赋予初始条件 $P_{-1}(x) = 0$ 和 $P_0(x) = 1$
2. 利用递推关系，构建 N 个在权函数 $w(x)$ 下正交的多项式
3. 求解每一个多项式的根
4. 根据式 6求解系数 $A_j$

## 2.4 实现

具体操作中调用了 **numpy.poly1d()** 类，简化了大部分计算过程。

```
import numpy as np
def sqrtbuild(N):
        '''
        构造以\sqrt[x]为权函数的最高位N的正交多项式
        '''
        res = [0 for _ in range(N+2)] #把 n =-1存在了第一个
        res[0] = np.poly1d([0])
        res[1] = np.poly1d([1])
        for x in range(1,N+1):
                p1 = np.poly1d([1,0])*res[x]**2
                p2 = res[x]**2
                p3 = res[x-1]**2
                a = sqrtinte(p1)/sqrtinte(p2)
                b = sqrtinte(p2)/sqrtinte(p3) if x >1 else 0
                res[x+1] = np.poly1d([1,-a])*res[x] - b*res[x-1] #防止出现0/0
        return res[1:len(res)]
```

求根直接调用 **poly1d** 的成员 $r$。

## 2.5 结果

表 2: 各非标准权函数的 Gauss 求积公式的求积系数和节点

| 机械求积公式 | $k$ | $x_k$ | $A_k$ |
|---|---|---|---|
| (3) | 0 | 0.28994920 | 0.27755600 |
| | 1 | 0.82116191 | 0.38911067 |
| (4) | 0 | -0.63245553 | 1.33333333 |
| | 1 | 0.63245553 | 1.33333333 |
| (5) | 0 | 0.16471029 | 0.12578267 |
| | 1 | 0.54986850 | 0.30760237 |
| | 2 | 0.90080583 | 0.23328162 |

## 2.6 结果检验

理论上来说，拥有 $n$ 个节点的 Gaussian 求积公式应当有 $2n+1$ 的代数精度。

对于 $n = 3$ 的情况，不难代入 mathematica 验证其精度与输入有效位数相同（即及其精度）

# 3 第三题：放射衰变问题

## 3.1 问题描述

考虑 A 和 B 两类原子核随时间的放射衰变问题，t 时刻，其布居数分别为 $N_A(t)$ 和 $N_B(t)$。假定 A 类核衰变为 B 类核，B 类核可以继续衰变，满足以下微分方程组：

$$\begin{aligned}\frac{dN_A}{dt} &= -\frac{N_A}{\tau_A}\\ \frac{dN_B}{dt} &= \frac{N_A}{\tau_A} - \frac{N_B}{\tau_B}\end{aligned}$$

其中,$\tau_A$ 和 $\tau_B$ 是时间衰变常数，在给定初始条件 $t_i = 0$ 时 $N_A(t_i) = N_B(t_i) = 1$ 下，回答下面三个问题：

1. 给出问题的解析解
2. 使用合适的算法数值求解上述耦合方程
3. 在给定 $\tau_A = 1s$，分别讨论 $\tau_B = 0.1s$,$1s$,$10s$, 三种情况下的短期和长期衰变行为。选取 $\tau_B = 10s$ 这种情况，讨论数值算法的误差，展示取不同步长 $\Delta t = 0.2s$,$0.1s$,$0.05s$ 时与解析结果的比较

## 3.2 解析结果

解析解可以直接给出：

$$\begin{aligned}N_A(t) &= e^{-t/\tau_A}\\ N_B(t) &= \frac{\tau_B}{\tau_A-\tau_B}e^{-t/\tau_A} + \frac{\tau_A-2\tau_B}{\tau_B-\tau_B}e^{-t/\tau_B}\end{aligned} \tag{7}$$

‘

## 3.3 思路

由于方程形式很简单，可以直接用有限差分的形式来代替微分, 从而得到递推关系：

$$\begin{aligned}N_A((i+1)\Delta t) - N_A(i\Delta t) &= -N_A(i\Delta t)\cdot\frac{\Delta t}{\tau_A}\\ N_B((i+1)\Delta t) - N_B(i\Delta t) &= -N_B(i\Delta t)\cdot\frac{\Delta t}{\tau_B} + N_A(i\Delta t)\cdot\frac{\Delta t}{\tau_A}\end{aligned}$$

经过 $N = t_{max}/\Delta t$ 步迭代就能得到答案。

## 3.4 实现

```
def decay(ta,tb,dt,tmax=1):
'''
衰变函数
返回<class='tuple'> (NA,NB)
'''
        n = 0
        NA=[1]
        NB=[1]
        while n*dt < tmax :
                NA.append(NA[n]*(1-dt/ta))
                NB.append(NB[n]*(1-dt/tb)+NA[n]*(dt/ta))
                n+=1
        return (NA,NB)
```

## 3.5 结果讨论

### 3.5.1 短期衰变

在固定 $\tau_A = 1s$ 后，可以认为，当时间 $t_{max}$ 小于任何一个半衰期，就可以认为是短期衰变，这里取固定 $t_{max} = 0.1s$

计算结果如下图: 不难看出，当 $t\tau_B = 0.1s < \tau_A$ 时，二者短程呈现下降

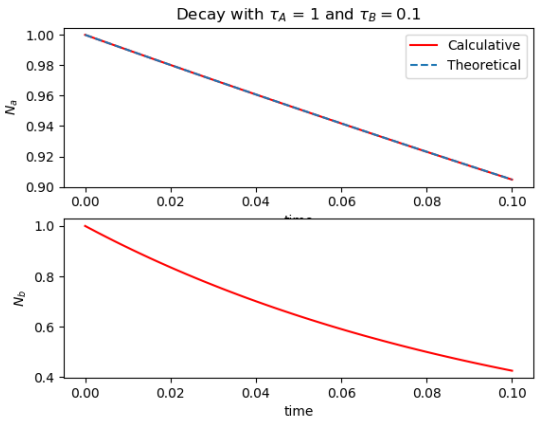



图 1: 短程结果

趋势。当 $\tau_B = 1 = \tau_A$ 时，$N_B$ 短程内几乎不发生变化。而当 $\tau_B = 10s > \tau_A$ 时，$N_B$ 开始呈现上升趋势，这与式 7是一致的。

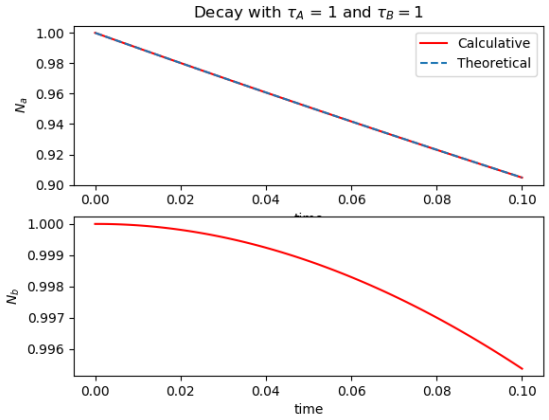



图 2: 短程结果

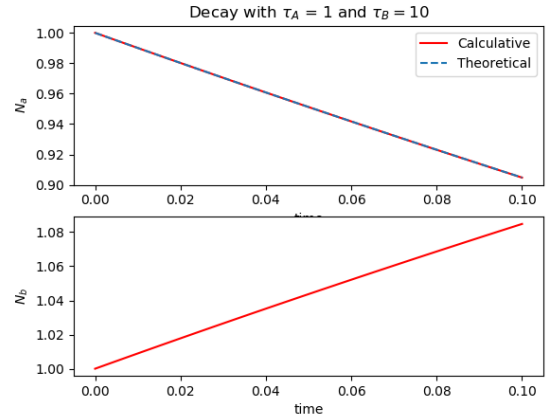



图 3: 短程结果

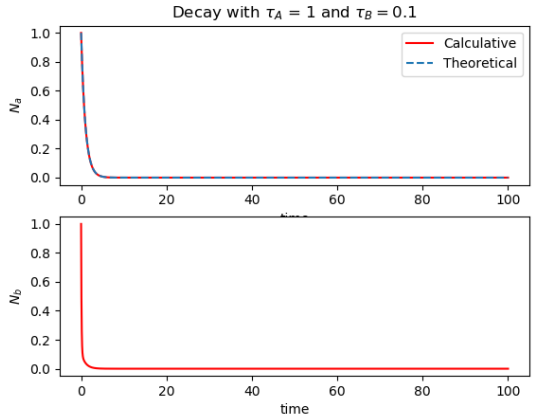



图 4: 长期衰变

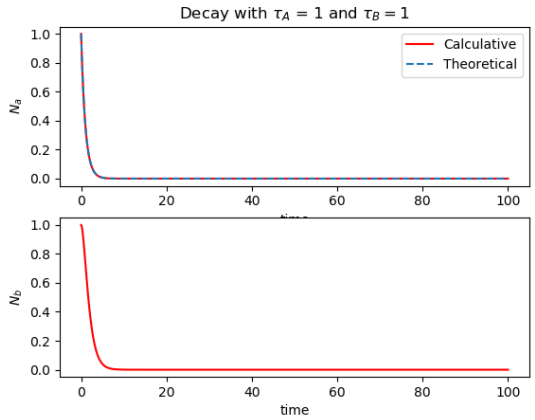



图 5: 长期衰变

### 3.5.2 长期衰变

同样地，我们取 $t_{max} = 100s > max(\tau_A, \tau_B)$，以研究长程情况

计算结果如下图： 也可以看出，去除最开始的那段短程过程有些许不同之外，在最后都会趋于零。

### 3.5.3 误差

对于每一次计算，可以利用泰勒展开估计误差

$$N((i+1)\Delta t) = N(i\Delta t) + N'(i\Delta t)\cdot\Delta t + \frac{1}{2}N^{(2)}(\xi)\Delta t^2$$

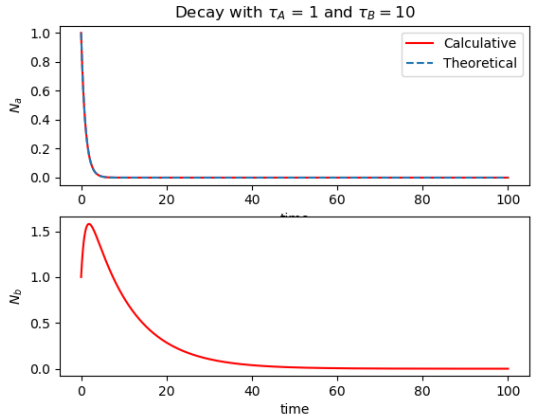



图 6: 长期衰变

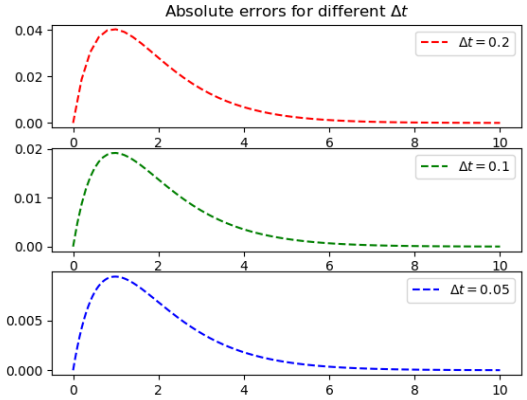



图 7: 误差随时间的变化

在忽略机器精度的情况下，绝对误差可以估计为一个上界：

$$\Delta(t) \leq \frac{1}{2}\frac{\Delta t \cdot t}{\tau^2}$$

等号当且仅当在 $t = 0$ 时成立。

下面是实际的情况（绝对误差随时间的变化），可以看出，在尺度上而言，的确是与 $\Delta t$ 的取值成正比

# 4 第四题：三次样条插值

## 4.1 问题描述

对于飞机机翼的轮廓每隔 $0.1m$ 的值，利用自然边界条件，使用三次样条插值估计任意一点的 $y$ 值，需要与原始数据点一同绘制出飞机的轮廓曲线。

| $x$ | 0 | 3 | 5 | 7 | 9 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|
| $y$ | 0 | 1.2 | 1.7 | 2.0 | 2.1 | 2.0 | 1.8 | 1.2 | 1.0 | 1.6 |

## 4.2 三次样条插值

三次样条差值也是多项式插值的一种。相比于拉格朗日插值和牛顿插值等多项式插值，三次样条方法拥有更加高阶的导数连续性。具体来说，三次样条方法不仅要求在插值点上有连续的函数值，而且还要求有连续的一阶以及二阶导数值。即因满足条件：

$$\begin{cases} S\left(x_i-0\right)=S\left(x_i+0\right) \\ S'\left(x_i-0\right)=S'\left(x_i+0\right), \quad i=1,2,\cdots,n-1 \\ S''\left(x_i-0\right)=S''\left(x_i+0\right) \end{cases}$$

在每一个区间 $x\in[x_k,x_{k+1}]$ 内部使用三次函数差值。

类似于 Hermite 插值法，使用分段 Hermite 多项式，此时零阶和一阶条件自动满足：

$$H_3^{(k)}(x)=y_k\alpha_0^{(k)}(x)+y_{k+1}\alpha_1^{(k)}(x)+y_k'\beta_0^{(k)}(x)+y_{k+1}'\beta_1^{(k)}(x)$$

取用各节点的一阶导数为未知数 $S'\left(x_i\right)=m_i$，利用二阶导数条件：

$$\mu_i m_{i+1}+2m_i+\lambda_i m_{i-1}=e_i, \quad i=1,2,\cdots,n-1$$

其中 $\mu_j=\frac{h_{j-1}}{h_{j-1}+h_j}, \quad \lambda_j=\frac{h_j}{h_{j-1}+h_j}, e_i=3\left(\lambda_i f\left[x_{i-1},x_i\right]+\mu_i f\left[x_i,x_{i+1}\right]\right)$

并且使用自然边界条件：

$$S''(0)=S''(x_{n-1})=0$$

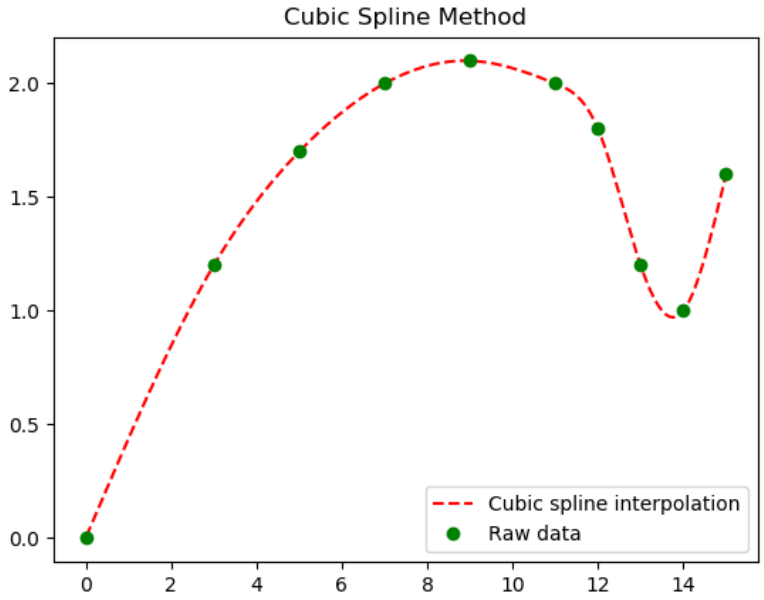
这样问题变为求解三对角矩阵方程解问题，可以利用追赶法快速求解。

## 4.3 实现

```
def firstorder(x,y):
        '''
        返回插值中n个点的一阶导数m_i
        采用三转角方程求解一阶导数
        '''
        # 建立矩阵
        mu=[h(i-1)/(h(i)+h(i-1)) for i in range(1,n)]
        lamb=[h(i)/(h(i)+h(i-1)) for i in range(1,n)]
        # 建立待解向量
        b = [3*d(0)]+[3*(lamb[i-1]*d(i-1)+mu[i-1]*d(i))\# 写不下换行
                for i in range(1,n)]+[3*d(n-1)]
        mu = [1] + mu
        lamb = lamb +[1]
        dia = [2 for _ in range(n+1)]
        # 开始追赶法求解三对角矩阵
        for i in range(1,n+1):
                lamb[i-1] = lamb[i-1]/dia[i-1]
                dia[i] = dia[i] - mu[i-1]* lamb[i-1]
                b[i] -= b[i-1]*lamb[i-1]
                m = [-1 for _ in range(n+1)]
        m[n] = b[n]/dia[n]
        for i in range(n-1,-1,-1):
                m[i] = (b[i] - mu[i]*m[i+1])/dia[i]
        return m
```

## 4.4 结果分析

可以看出，函数的光滑性是很好的。



# 5 第五题：求解对称带状正定矩阵

## 5.1 题目描述

对于带宽为 $2m+1$ 的对称正定带状矩阵 $\mathbf{A}$, 可以分解为 $\mathbf{A}=\mathbf{LDL}^T$。结果可以用于求解一般的对称正定带状线性方程组 $\mathbf{Ax}=\mathbf{b}$。

题目中给定的对称带状矩阵 $\mathbf{A}$,$m=2$, 非零元素为 $a_{11}=a_{nn}=5$; $a_{ii}=6(i=2,\ldots,n-1)$; $a_{i,i-1}=4(i=2,\ldots,n);a_{i,i-2}=1(i=3,\ldots,n)$，右端向量为 $\mathbf{b}=(60,120,\ldots,120,60)^T$。给出 $n=100$ 和 $n=10000$ 两个算例。

## 5.2 算法描述

对称带状矩阵也可以使用 Cholesky 分解方法, 可以假定 $\mathbf{D}_{ij}=1/l_{ii}$，由矩阵乘法的可以得到公式：

$$l_{ij}=a_{ij}-\sum_{k=r}^{j-1}l_{ik}l_{jk}/l_{kk},i=0,1,2,\ldots,n-1$$

其中：

$$r=\begin{cases}0, & i\le m+1\\ i-m, & i>m+1\end{cases}$$

同时将右端向量也做分解,$\mathbf{b}=\mathbf{LD}\tilde{\mathbf{b}}$，具体来说：

$$\tilde{b}_i=b_i-\sum_{j=r}^{i-1}l_{ij}\tilde{b}_j/l_{jj},\quad i=0,1,2,\ldots,n-1$$

最后再求解等价线性方程组，$\mathbf{L^Tx}=\tilde{\mathbf{b}}$, 利用回代过程:

$$x_i=\left(\tilde{b}_i-\sum_{j=i+1}^{t}l_{ji}x_j\right)/l_{ii},i=n-1,\ldots,1,0$$

其中:

$$t=\begin{cases}n-1, & i>n-m-1\\ i+m, & i\le n-m-1\end{cases}$$

关于储存，由于题干中要求时空复杂度做到最小，因此我们采用 $(m+1)$ 个向量（数组）来储存，在求解的过程中不需要额外的向量来储存矩阵元。解向量也只用一个 $n$ 维向量储存，不需要额外的空间。总的来说，空间复杂度为 $\mathcal{O}(mn)$。在时间复杂度方面，第一步 Cholesky 分解的复杂度为 $\mathcal{O}(m^2n)$，而后的每一步复杂度都为 $\mathcal{O}(mn)$。总的来说，计算的空间复杂度为 $\mathcal{O}(m^2n)$

## 5.3 实现

```
def cholesky(n, m, a, b):
        """
        Cholesky求解方法,返回解向量和矩阵
        return a,b
        """
        # print(a)c
        for k in range(n):
                #先解对角元素
                a[0][k] -= sum([a[k-j][j]**2*a[0][j] for j in range(max(0,k-m),k)])
                for l in range(1, min(n-k,m+1)):
                        a[l][k] -= sum([a[k-j][j]*a[0][j]*a[l+k-j][j] for j in\
                        range(max(0,l+k-m),k)])
                        a[l][k] /= a[0][k]
        #回代解方程
        for k in range(n):
                b[k] -= sum([a[k-j][j]*b[j] for j in range(max(0,k-m),k)])
        for k in range(n):
                b[k] /= a[0][k]
        for k in range(n):
                ki = n-1-k
                b[ki] -= sum([a[j][ki]*b[ki+j] for j in range(1,min(n-ki,m+1))])
        return a,b
```

## 5.4 结果讨论

为了更加直观地讨论问题，将解出来的解向量画为折线图，可以看出，体系包络的大小正比于 $n^2$

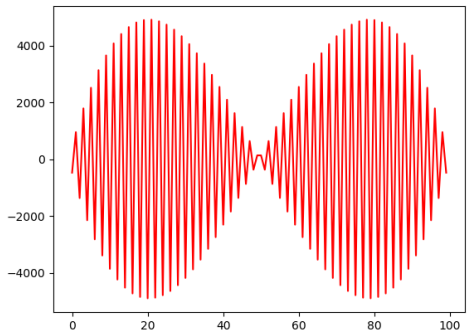



图 8: $n = 100$ 算例

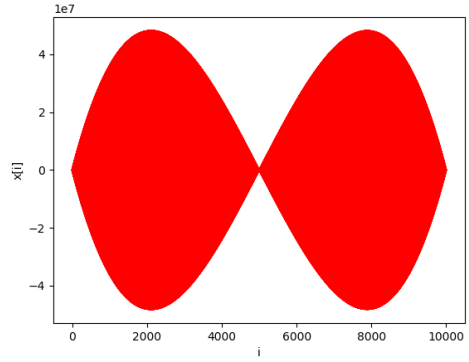



图 9: $n = 10000$ 算例

# 6 第六题：共轭梯度法求解稀疏矩阵线性方程组

## 6.1 题目要求

对于给定的稀疏矩阵

$$\mathbf{A} = \begin{bmatrix} 3 & -1 & 0 & \cdots & 0 & \frac{1}{2} \\ -1 & 3 & -1 & \cdots & \frac{1}{2} & 0 \\ 0 & -1 & 3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & \frac{1}{2} & 0 & \cdots & 3 & -1 \\ \frac{1}{2} & 0 & 0 & \cdots & -1 & 3 \end{bmatrix}$$

利用共轭梯度法, 求解线性方程组 $\mathbf{Ax} = \mathbf{b}$，并且要求两次的迭代误差要求小于 $\epsilon = 10^{-6}$。

分别取 $n = 100$ 和 $n = 10000$ 展示结果。

## 6.2 算法描述

共轭梯度法本质上属于一种迭代法，因此可以很好地利用矩阵稀疏性。相比于最速下降法，共轭梯度法避免了每一次下降的方向不变的缺点。实际上，每一步下降的矢量方向 $p_i$ 都是 $\mathbf{A}$ 正交的，即 $p_i^T \mathbf{A} p_j = 0$

由于题目中给定的矩阵是一个对称正定矩阵，因此不用进一步进行预处理即可直接应用共轭梯度法。

算法判停的条件是

$$\|\mathbf{b} - \mathbf{A}\mathbf{x_k}\| / \|\mathbf{b}\| \leq \epsilon$$

具体操作上是使用的 2– 范数

## 6.3 实现

```python
def CGM(e,b,A,x=[0 for _ in range(n)]):
        '''Input error bound e and ndarray b and sparse matrix A
        return solution ndarray x
        '''
        x = np.array(x)
        r = b - A*x # 计算初始残差
        p = r # 计算初始方向矢量
        epsilon = np.dot(r.T,r)
        k = 0
        while epsilon >= e :
                delt = A*p
                a = np.dot(r,p)/np.dot(r,delt) # a是标量
                x = x + a*p
                r = r - a*delt # 更新残差
                beta = - np.dot(r,delt)/np.dot(p,delt)
                p = r + beta*p # 更新下降方向
                k += 1
                epsilon = np.dot(r.T,r) # 计算误差矢量的2–范数
                print(k,"error : ",epsilon) #输出每一步的数值
        return x
```

## 6.4 结果讨论

直接画出结果随坐标的变化 可以发现，结果残差被控制得很好。

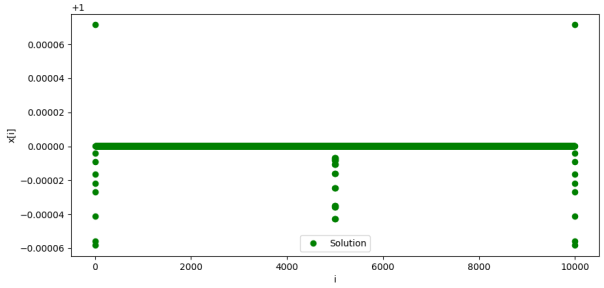



图 10: $n = 10000$ 的算例

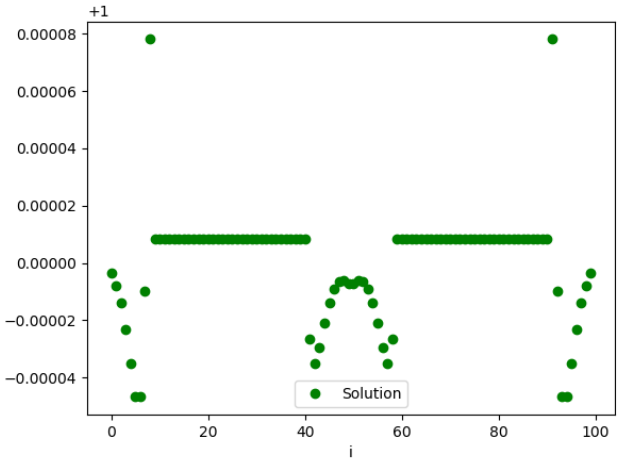



图 11: $n = 100$ 的算例

# 7 第七题：氢原子定态波函数求解

## 7.1 广义拉盖尔多项式

### 7.1.1 原理

氢原子的径向波函数可以表示为广义拉盖尔多项式：

$$R_{nl}(r) = \sqrt{\left(\frac{2}{na_\mu}\right)^3 \frac{(n-l-1)!}{2n(n+l)!}} e^{-r/na_\mu} \left(\frac{2r}{na_\mu}\right)^l L_{n-l-1}^{2l+1}\left(\frac{2r}{na_\mu}\right)$$

对此，要求解氢原子波函数问题，就需要计算广义拉盖尔多项式 $L_n^\alpha(x)$。

具体来说，可以利用广义拉盖尔的递推公式：

$$L_{n+1}^{(\alpha)}(x) = \frac{1}{n+1}\left((2n+1+\alpha-x)L_n^{(\alpha)}(x) - (n+\alpha)L_{n-1}^{(\alpha)}(x)\right)$$

以及初始条件：

$$L_0^{(\alpha)}(x) = 1$$
$$L_1^{(\alpha)}(x) = -x + \alpha + 1$$

这样，就可以直接通过递推得到广义拉盖尔多项式的值。时间复杂度为$\mathcal{O}(n)$。

### 7.1.2 实现

```
def LaguerreL(n,a,x):
        '''
        To calculate generalized Laguerre polinomial
        return L^a_n(x)
        '''
        Li = 1
        Lip = 1-x+a
        if n == 1:
                return Lip
        elif n==0:
                return Li
        else:
                for i in range(1,n):# 进行迭代
                        Li,Lip=Lip,1/(i+1)*((2*i+1+a-x)*Lip-(i+a)*Li)
        return Lip
```

### 7.1.3 结果

注意到 Python 的 **Scipy** 库中有 **scipy.special.genlaguerre** 方法，因此利用内置函数计算出这种方法的误差：

以及下面为题目中要求的表格：

| $x = 10^{-3}$ | n=3 | 10 | 30 |
|---|---|---|---|
| $\alpha = 2$ | 9.9900 | 65.780 | 491.058 |
| 20 | 1770.747 | 30030710.776 | 47061929154802.58 |
| 40 | 12340.097 | 10269773004.722 | 5.5307e+19 |

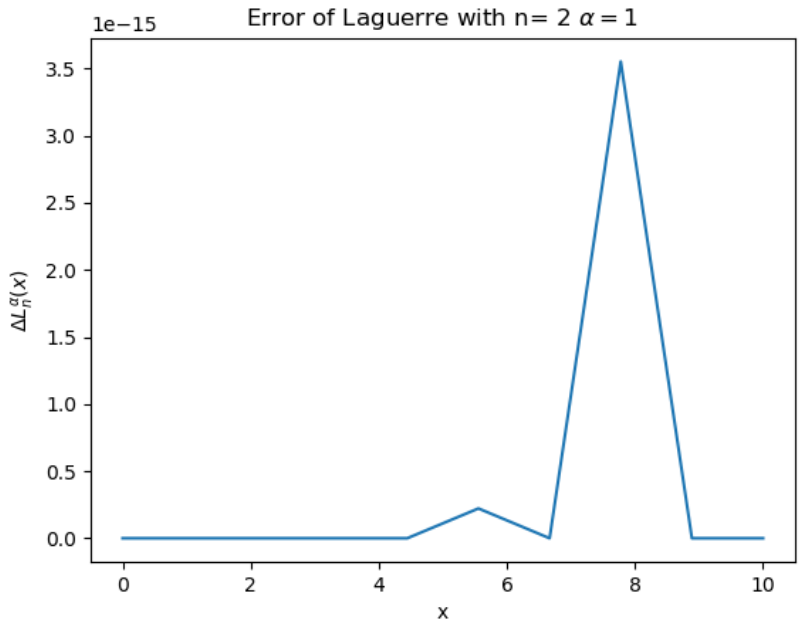



图 12: 广义拉盖尔函数与内置函数的误差

## 7.2 球谐函数

### 7.2.1 思路

求解球谐函数的关键是求解连带勒让德多项式，同样地我们采用递推公式方法。

$$(\ell - m + 1)P_{\ell+1}^m(x) = (2\ell + 1)xP_\ell^m(x) - (\ell + m)P_{\ell-1}^m(x)$$
$$2mxP_\ell^m(x) = -\sqrt{1 - x^2}\left[P_\ell^{m+1}(x) + (\ell + m)(\ell - m + 1)P_\ell^{m-1}(x)\right]$$

实际上我们已知的是 $P_0^0(x) = 1$ 和 $P_1^0(x) = x$。我们的目的可以变为在一个二维网格上从 $(0, 0)$ 点走到 $(l, m)$ 点。思路是首先利用勒让德多项式的递推公式走到 $(l, 0)$ 点，再利用连带勒让德的递推公式走到 $(l, m)$ 点。

### 7.2.2 实现

```
1 def GeneralizedLegrend(l,m,x):
2         '''
3         求解连带勒让德函数
4         基本思路是利用递推公式达到二维平面上(l,m)这个点
```

```
        先达到(l,0)，求解勒让德函数
        返回连带勒让德P^l_m(x)
        '''
        Pi = 1
        Pip = x
        sq = ms.sqrt(1-x**2) #减少开方运算
        for i in range(1,l):
            Pip,Pi=((2*i+1)*x*Pip-i*Pi)/(i+1),Pip
        # 利用(l,0)和(l-1,0)求出(l,1)
        Pii = l*(x*Pip-Pi)/sq
        # 继续迭代得到(l,m)
        for i in range(1,m):
            Pii,Pip=(-2*i*x*Pii/sq-(l+i)*(l-i+1)*Pip),Pii
        return Pii
```

### 7.2.3 问题

上面这种朴素的方法能够在 $l$ 和 $m$ 很小的时候达到很高的精度，但是对于作业中要求的 $l=500$ 和 $m=499$ 的问题，会出现上溢问题，因此采用重新定义连带勒让德函数的方式来解决。

定义新的连带勒让德函数：

$$\widetilde{P}_l^m(x)=\sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(x)$$

满足新的递推关系：

$$\widetilde{P}_l^m(x)=x\sqrt{\frac{4l^2-1}{l^2-m^2}}\widetilde{P}_{l-1}^m(x)-\sqrt{\frac{2l+1}{2l-3}\frac{(l-1)^2-m^2}{l^2-m^2}}\widetilde{P}_{l-2}^m(x)$$

注意到此时的 $\widetilde{P}_l^m$ 满足的递推关系中系数都是接近于 1 的数，不至于导致严重的下溢和上溢。

以及通过连带勒让德函数的定义不难看出：

$$\widetilde{P}_m^m(x)=(-)^m\frac{(1-x^2)^{m/2}}{2^m m!}(2m)!\sqrt{\frac{2m+1}{4\pi(2m)!}}=(-)^m\left(1-x^2\right)^{m/2}\sqrt{\frac{(2m+1)!!}{4\pi(2m)!!}}$$

以及有定义 $\widetilde{P}_l^m(x)=0$ 当 $m<l$ 时。这样就不难从递推公式和初始条件中计算出所有的连带勒让德函数。而且我们注意到实际上 $\widetilde{P}_l^m(x)$ 和 $\widetilde{P}_m^m(x)$ 是成正比的，即：

$$\widetilde{P}_l^m(x) = f(l,m,x)\widetilde{P}_m^m(x)$$

因此实际计算中可以先不代入 $\widetilde{P}_m^m(x)$，以减少溢出风险。为了进一步减少溢出风险，采用对数输出形式，即:

$$\log_{10}|\widetilde{P}_l^m(x)| = \log_{10}|f(l,m,x)| + \log_{10}|\widetilde{P}_m^m(x)|$$

虽然牺牲了精度，但是极大减少了溢出风险。

### 7.2.4 实现

```
def New_GeneralizedLegrend(l,m,x):
        '''
        新的连带勒让德函数计算方法，减少溢出风险
        '''
        P = 0
        Pi = 1# 并不直接代入P_m^m(x)
        if l-m< 0:
                return 0
        for a in range(m+1,l+1):
                Pi,P=\
                x*ms.sqrt((4*a**2-1.0)/(a**2-m**2))*Pi\
                -ms.sqrt((2*a+1.0)/(2*a-3.0)*\
                ((a-1.0)**2-m**2)/(a**2-m**2))*P , Pi
        d = 1
                for a in range(1,m+1):
                d *= (2*a+1)/(2*a)
        res = ms.log10(abs(Pi))+m/2.0*ms.log10(1-x**2)+\
        1/2.0*ms.log10(d/(4*ms.pi))
        return res
```

### 7.2.5 结果

结果储存在文件**./data/Harmonic.txt** 之中。

值得注意的是，当 $L = 1000, M = 999$ 时，$Log_{10}(|Y_{lm}|)$ 已经达到了 $10-2498$ 量级

## 7.3 电子密度

### 7.3.1 思路

由于之前已经得出了球谐函数和拉盖尔函数的计算方法，下面直接调用朴素的连带勒让德函数。

### 7.3.2 结果

注意到归一化系数实际上对画热力图没有影响，故下面的图中都略去了归一化系数，绝对数值是不可靠的。而且考虑到 $m$ 仅对角向有贡献，而实际上二维图应该是各项同性的，因此只放 $m = 0$ 的图。

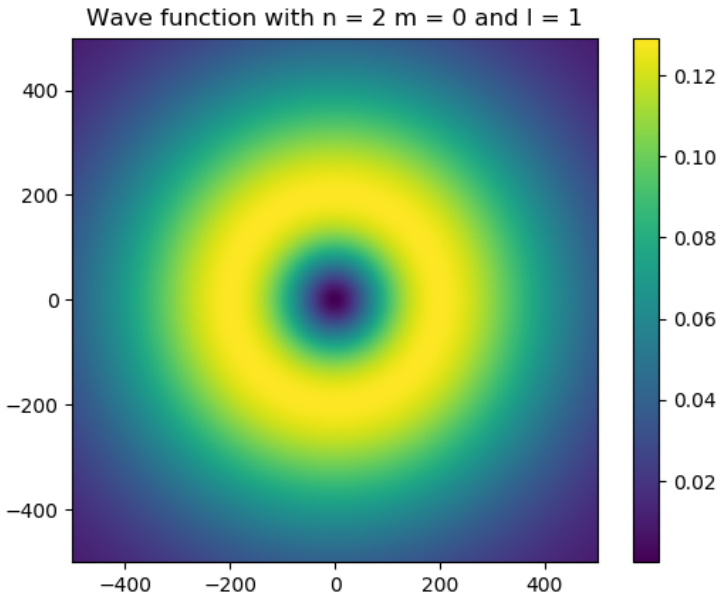



图 13: 氢原子波函数图

## 7.4 计算氢原子能级

### 7.4.1 思路

由于氢原子的 $SO(4)$ 对称性，氢原子能级与磁量子数 $m$ 无关：

$$\begin{aligned} E_{nl} &= \langle \psi_{nl} | \hat{H} | \psi_{nl} \rangle \\ &= -\langle \psi_{nl} | \frac{1}{2r^2} \frac{\partial}{\partial r} \left( r^2 \frac{\partial}{\partial r} \right) + \frac{l(l+1)}{2r^2} + \frac{1}{r} | \psi_{nl} \rangle \end{aligned}$$

对于题目中给定的 $n = 3, l = 1, m = 1$ 的波函数

$$\begin{aligned} \Psi(r, \theta, \varphi) &= \sqrt{\left(\frac{2}{3}\right)^3 \frac{(n-l-1)!}{2n(n+l)!}} \mathrm{e}^{-\rho/2} \rho^l L_{n-l-1}^{2l+1}(\rho) Y_l^m(\theta, \varphi) \\ &= \frac{4}{81\sqrt{6}} \mathrm{e}^{-r/3} r(6-r) Y_1^1(\theta, \varphi) \end{aligned}$$

设 $f = \mathrm{e}^{-r/3} r(6-r)$，代入 $l = 1$，并且取积分域为 $[0, 60)$，得到能量的积分表达式。

$$E = \frac{8}{3^9} \int_0^{60} f \left( f(1-r) - rf' - \frac{r^2 f''}{2} \right) \mathrm{d}r$$

在求解过程中通过不停减小积分间隔以达到精度要求。

### 7.4.2 实现

```
def energy():
    '''
    能量函数，计算氢原子能量
    '''
    epsi = 10**(-6)# 数值精度
    d =60.0 # 数据范围
    n = 2 # 初始分隔
    S = d/n*(y(0)+y(d))
    while abs(S*8/3**9+1/18.0) > epsi:
        n *= 2
        S = sum([y(d*i/(n-1)) for i in range(0,n)])/(n/d)
        print('n = '+str(n))
        print('Error = '+str(S*8/3**9+1/18.0))
    return S*8/3**9

def y(r):
```

```
17          h = 0.001 #微分精度
18          k = f(r)
19          dy = (f(r+h)-f(r-h))/(2*h)
20          ddy = (f(r+h)+f(r-h)-2*f(r))/h**2
21          return k*(k*(1-r)-r*dy-r**2*ddy/2)
22
23  def f(r):
24          return ms.exp(-r/3.0)*r*(6-r)
```

### 7.4.3 结果讨论

数值计算在 $n = 65536$ 处停止，数值精度为 $e = 8.5 \times 10^{-7}$，具体结果如下图:

```
n = 65536
Error = 8.465007297411842e-07
-0.05555470905482581
```

图 14: 数值计算氢原子能级结果