CompPhys Assignment 03

李尚坤 物理学系 20307130215

1 证明时间复杂度

1.1 题目描述

Prove that the time complexity of the Gaussian elimination algorithm is $O(N^3)$.

1.2 高斯消元法步骤

以一个 N 元一次方程组为例, 其对应的系数矩阵为一个 N×N 的方阵, 增广矩阵为一个 N×(N+1) 的矩阵, 高斯消元法的步骤如下。

1. 首先高斯消元法的行操作一共有 3 类: 交换两行, 将一行乘以某非零数, 将一行乘以某数后加到另一行去;
2. 高斯消元法首先将第一行（总可以通过交换使得第一行第一个元素非零）乘以某数加到第二行, 使得第二行第一个元素为零, 同理对第三行、第四行进行操作, 使得第一列除了第一行之外 (第一主元) 全部元素为零;
3. 对第二行进行类似操作, 最终使得第二列第二主元以下的全部元素为零;
4. 依次对第三行、第四行一直到第 N 行进行操作, 最终达到高斯消元的目的。

1.3 证明

接下来我们分析这一算法的时间复杂度。

在上述步骤 2 中, 第一行有 N 个元素, 将第一行乘以某个数加到另一行去, 一共进行了 N 次乘法,N 次加法。因为一共需要对 (N−1) 行进行操作, 因此可以认为步骤 2 一共需要进行 2N×(N−1) 次运算。

同理, 在上述步骤 3 中需要对第二行以下的各行进行处理. 每处理一行, 需要进行了 N−1 次乘法,N−1 次加法, 需要处理的行数为 N−2。因此, 此步骤一共需要进行 2(N−1)×(N−2) 次运算。

综上, 我们一共需要进行的运算次数为

$$S = 2N(N-1) + 2(N-1)(N-2) + \cdots + 2\times 2\times 1 + 2\times 1\times 0 \tag{1}$$

由于:

$$N^2+(N-1)^2+\cdots+2^2+1^2=\frac{1}{3}N(N+\frac{1}{2})(N+1) \tag{2}$$

$$N+(N-1)+\cdots+2+1=\frac{1}{2}N(N+1) \tag{3}$$

故 $S=\frac{2}{3}N(N+\frac{1}{2})(N+1)-N(N+1)$, 当 N 很大时,$S\sim N^3$, 因此高斯消元法的时间复杂度为 $O(N^3)$。

2 计算 n×m 矩阵的 RREF 型

2.1 题目描述

Write a general code to transform a n×m matrix into the REDUCED ROW ECHELON FORM, and use the code to obtain the RREF of the following matrix.

$$\begin{bmatrix} 2 & 8 & 4 & 2 \\ 2 & 5 & 1 & 5 \\ 4 & 10 & -1 & 1 \end{bmatrix}$$

2.2 解决方案描述

考虑如下矩阵，我们先将第 1 列的各元素 a_{i1} 进行比较，将最大值所在的行换到第 1 行。然后将第 1 行每个元素除以 a_{11}，变成下面右侧的矩阵。

$$\begin{bmatrix} a_{11} & a_{12} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{21} & a_{21} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a_{12}^{'} & a_{12}^{'} & \dots & a_{1m}^{'} \\ a_{21} & a_{21} & a_{21} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix}$$

随后将第 1 行乘以某数之后加到后面各行，使得第 1 列中 a_{11} 以下各元素全部为 0, 如下左侧矩阵所示。如上所述，我们对第 2 行、第 3 行 ... 第 n 行进行类似的操作，最终可以得到下方右侧所示的上三角矩阵。

$$\begin{bmatrix} 1 & a_{12}^{'} & a_{12}^{'} & \dots & a_{1m}^{'} \\ 0 & a_{21}^{'} & a_{21}^{'} & \dots & a_{2m}^{'} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & a_{n2}^{'} & a_{n3}^{'} & \dots & a_{nm}^{'} \end{bmatrix} \rightarrow \begin{bmatrix} 1 & a_{12}^{'} & a_{12}^{'} & \dots & a_{1m}^{'} \\ 0 & 1 & a_{21}^{''} & \dots & a_{2m}^{''} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

再从最后一行开始，对上方右侧的上三角矩阵重复类似的步骤，将主元上方的元素全部化为 0，即可得到矩阵的最简行阶梯形（RREF）。

2.3 伪代码

Algorithm 1: Calculate RREF

Input: The N×M Matrix R needed to be calculated.

Output: The RREF of the initial Matrix.

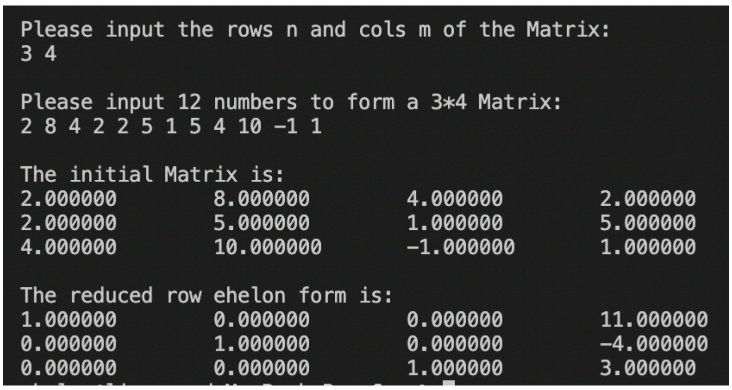
for $i \leftarrow 1$ **to** n **do**
    $m \leftarrow$ the row index of $max(a_{ij}, j \in (1, n))$
    swap row i and row m of the R
    $R_i \leftarrow R_i / R_{ii}$ //Let the first element of row i equals 0
    for $j \leftarrow i + 1$ **to** n **do**
        $R_j \leftarrow R_j - R_i * R_{ji}$
    end
end
for $i \leftarrow n$ **to** 1 **do**
    for $j \leftarrow i - 1$ **to** 1 **do**
        $R_j \leftarrow R_j - R_i * R_{ji}$
    end
end
return R

2.4 输入/输出示例

1. 对于需要计算的矩阵（下方左侧矩阵），首先输入需要计算的矩阵的行数和列数，再依次输入元素，计算结果为下方右侧矩阵。

$$\begin{bmatrix} 2 & 8 & 4 & 2 \\ 2 & 5 & 1 & 5 \\ 4 & 10 & -1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 11.000000 \\ 0.000000 & 1.000000 & 0.000000 & -4.000000 \\ 0.000000 & 0\ 000000 & 1\ 000000 & 3.000000 \end{bmatrix}$$

2. 程序在终端运行时的输入/输出截图如下：



```
Please input the rows n and cols m of the Matrix:
3 4

Please input 12 numbers to form a 3*4 Matrix:
2 8 4 2 2 5 1 5 4 10 -1 1

The initial Matrix is:
2.000000        8.000000        4.000000        2.000000
2.000000        5.000000        1.000000        5.000000
4.000000        10.000000       -1.000000       1.000000

The reduced row ehelon form is:
1.000000        0.000000        0.000000        11.000000
0.000000        1.000000        0.000000        -4.000000
0.000000        0.000000        1.000000        3.000000
```

图 1: 求解矩阵的 RREF

2.5 用户手册

1. 本程序的源程序为 CalRREF.cpp, 可执行文件为 CalRREF.exe
2. 为了防止直接运行.exe 文件时 cmd 在运行结束后直接关闭，在源文件的末尾增加了一行命令 system(“pause”)
3. 本程序使用高斯消元法对矩阵进行化简，最终将矩阵化为最简行阶梯形
4. 运行程序后，在终端首先跳出提示语“Please input the rows n and cols m of the Matrix:”，随后分别输入两个整数，以空格隔开，作为矩阵的行数和列数
5. 输入完毕后按下回车，再次跳出提示语“Please input n*m numbers to form a n*m Matrix:”，按顺序输入 n*m 个数后，按回车键即可得到最终结果。注意，输入时每个数之间也用空格隔开

3 求解一维定态薛定谔方程

3.1 题目描述

Solve the 1D Schrodinger equation with the potential (i) $V(x) = x^2$; (ii) $V(x) = x^4 - x^2$ with the variational approach using a Gaussian basis (either fixed widths or fixed centers). Consider the three lowest energy eigenstates.

3.2 解决方案描述

在本题中我们利用高斯基来求解一维定态薛定谔方程。高斯基有两个变量，结果显示当固定宽度 ν，调整中心 s 时可以获得比较满意的解。

因此，我们选择固定 $\nu = 0.5$，调整 s 的值，通过积分

$$H_{ij} = \int_{-\infty}^{+\infty} \phi_i^*(\nu, s_i, x)\hat{H}\phi_j(\nu, s_j, x)dx \quad S_{ij} = \int_{-\infty}^{+\infty} \phi_i^*(\nu, s_i, x)\phi_j(\nu, s_j, x)dx$$

计算出对应的哈密顿矩阵 H 和矩阵 S, 然后求解广义本征值问题 $HC = ESC$, 即可得到能量的可能取值。

本题中，对于势能 $V(x) = x^2$ 与 $V(x) = x^4 - x^2$ 对应的 H_{ij} 与 S_{ij} 的积分计算在 Mathematica 中完成，具体计算过程参见 SolSchor.nb 文件。

3.3 伪代码

Algorithm 2: Solve Schrodinger equation

Input: The potential $V(x)$ and the number of Gaussian basis n

Output: The three lowest energy eigenstates.

genarate Matrix $H[n][n]$ and $S[n][n]$
for $i \leftarrow 1$ **to** n **do**
  $s_i \leftarrow -(n-1)/4 + 0.5 * i$
  for $j \leftarrow 1$ **to** n **do**
    $s_j \leftarrow -(n-1)/4 + 0.5 * j$
    $H[i,j] \leftarrow \int_{-\infty}^{+\infty} \phi_i^*(\nu, s_i, x)\hat{H}\phi_j(\nu, s_j, x)dx$
    $S[i,j] \leftarrow \int_{-\infty}^{+\infty} \phi_i^*(\nu, s_i, x)\phi_j(\nu, s_j, x)dx$
  end
end
$eigvals \leftarrow$ EIGENVALUES(H,S) //solve the generalized eigenvalue problem
$E \leftarrow$ SORT($eigvals$)
PLOT the wave function of the three lowest eigenvalues
return $E[1:3]$ and the wave function of three eigenvalues

3.4 输入/输出示例

1. 对于本题，我们分别输入不同的高斯基的个数，得到的输出如下表 1、表 2 所示：

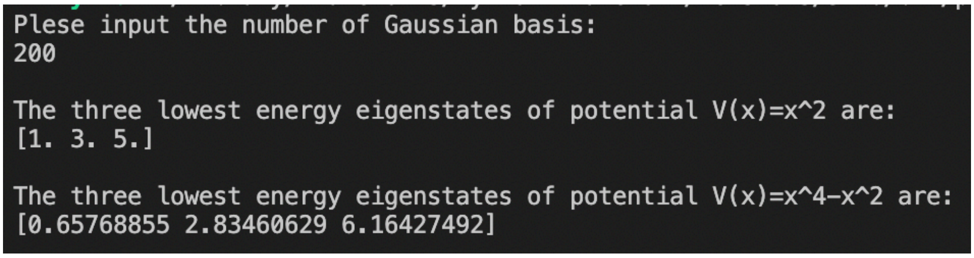
Input n	Output E	Note
100	1.000000, 3.000000, 5.000000	我们可以看见这三个值与理论是符合的（谐振子势能）
200	1.000000, 3.000000, 5.000000	
300	1.000000, 3.000000, 5.000000	

表 1: $V(x) = x^2$ 输出结果

Input n	Output E	Note
100	0.657662, 2.834569, 6.164021	可以看见取不同的基，程序的稳定性很好
200	0.657689, 2.834606, 6.154275	
300	0.657632, 2.831907, 6.163549	

表 2: $V(x) = x^4 - x^2$ 输出结果

2. 程序在终端运行时的输入/输出截图如下：



```
Plese input the number of Gaussian basis:
200

The three lowest energy eigenstates of potential V(x)=x^2 are:
[1. 3. 5.]

The three lowest energy eigenstates of potential V(x)=x^4-x^2 are:
[0.65768855 2.83460629 6.16427492]
```

图 2: 求解一维定态薛定谔方程

3. 程序输出的函数图像如图 3，图 4 所示：

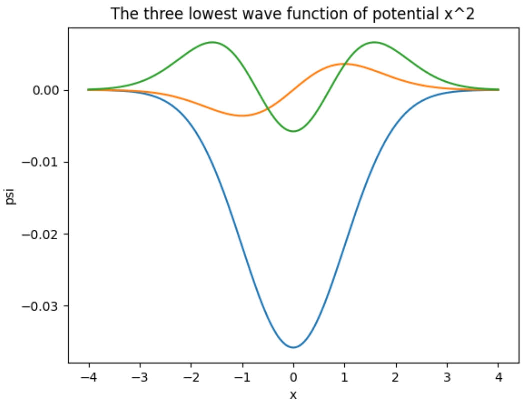


图 3: $V(x) = x^2$

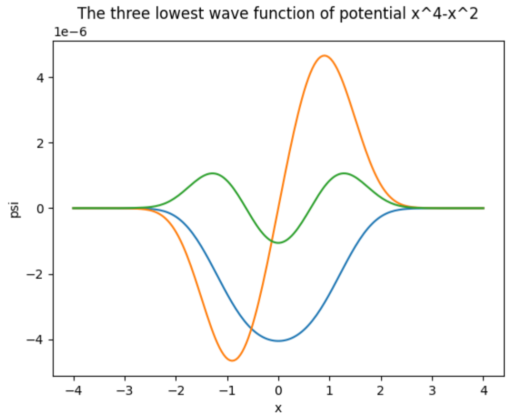


图 4: $V(x) = x^4 - x^2$

3.5 用户手册

1. 本程序的源程序为 SolShor.py
2. 在执行源程序之前，应当先安装 numpy、scipy 以及 Matplotlib 库
3. 本程序利用高斯基，固定高斯基函数中的 ν，改变函数的中心 s，求解一维定态薛定谔方程
4. 运行程序后，在终端首先跳出提示语“Plese input the number of Gaussian basis:”，输入一个整数，指定题目中求解时使用的高斯基的个数
5. 输入完毕后按下回车，会在终端输出势能 $V(x) = x^2$ 与 $V(x) = x^4 - x^2$ 最低的三个能量值
6. 同时，首先展示势能 $V(x) = x^2$ 对应的波函数图像，关闭这一窗口后，展示势能 $V(x) = x^4 - x^2$ 对应的波函数图像