计算物理作业 3

杨远青　22300190015　CompPhys 24

2024 年 9 月 28 日

远方来朋，喜；假期俱至，悦。

1　~~题目 1：~~高斯消元法的时间复杂度分析

1.1　题目描述

Prove that the time complexity of Gaussian elimination algorithm is $\mathcal{O}(n^3)$.

1.2　证明

Gaussian 消元法，此处特指 *Forward Elimination* & *Backward Substitution* 法，而不是最古老的 Gaussian-Jordan 消元法（用于求逆的某浪漫主义教学算法），在大多数情况下的表现，并不如兼具精确度与效率的 $\boldsymbol{LU}$ 分解法，但一些思想被嵌入后者与适用于更大规模矩阵求解的各类迭代算法中，因此仍有必要对其进行分析。

先考虑 *Forward Elimination* 的时间复杂度，即通过初等行变换将原本的增广矩阵 $(\boldsymbol{A}\,|\,\boldsymbol{b})$

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right]$$

上三角化为 $\boldsymbol{U}$。暂不考虑 *Pivot* 步骤可能带来的交换操作，尽管这对于提升数值稳定性非常重要。考虑第 1 列的第 2 至 n 行，每一行需要先计算系数 a_{i1}/a_{11}，再进行 n 次乘法与 n 次减法（各行首元素直接设为 0，不计入乘减法操作，但要考虑最右侧 b 的元素），故第 1 列的消元操作数为 $(n-1)(2n+1)$，递推可知，第 i 步便是对 $(n-i+1)\times(n-i+1)$ 子矩阵的消元，迭代操作数为 $(n-i)(2n-2i+3)$，总操作数为

$$T_F(n)=\sum_{i=1}^{n-1}(2n-2i+3)(n-i)=2\sum_{i=1}^{n-1}(n-i)(n-i)+3\sum_{i=1}^{n-1}(n-i)=\frac{4n^3+3n^2-7n}{6}.$$

再考虑 *Backward Substitution* 的时间复杂度，当我们消元得到一个 $n\times n$ 的上三角矩阵 $\boldsymbol{U}$

$$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} & b'_n \end{array}\right]$$

之后，需要从最后一行开始，逐行求解

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{k} a'_{ij} x_j \right).$$

每一行涉及的四则运算（我们非常流氓地忽视除法的独特地位，理论上这需要基于牛顿迭代的现代方法进行特殊处理）为 $(n-i)$ 次乘法与 $(n-i)$ 次减法，再进行 1 次除法，故每行的操作数为 $[2(n-i)+1]$，总操作数为

$$T_B(n) = \sum_{i=1}^{n} [2(n-i)+1] = 2\sum_{i=1}^{n}(n-i) + n = n^2.$$

故 Gaussian 消元法的总操作数为

$$T(n) = T_F(n) + T_B(n) = \frac{4n^3+3n^2-7n}{6} + n^2 = \frac{4n^3+9n^2-7n}{6}.$$

其中有除法 $n(n+1)/2$ 次，乘法与减法各 $n(n-1)(2n+5)/6$ 次，故

$$\boxed{T(n) = \mathcal{O}(n^3)}$$

伙计，这听起来一点也不酷，怎么到头来还是和求逆矩阵一样是 $\mathcal{O}(n^3)$? 但如果我们将 *Substitution* 的思想嵌入到 $\boldsymbol{LU}$ 分解法[1]，对一些特定情形，譬如三对角矩阵的回代操作可以从 $\mathcal{O}(n^2)$ 优化到 $\mathcal{O}(n)$，且对于不同的待解向量 $\boldsymbol{b}$，我们的圣遗物 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 可以被重复利用，这听上去还是不错的！

如果想和理论计算机科学家一样，执着于对 $\mathcal{O}(n^3)$ 的优化：Strassen 的构造可以帮你将指数因子优化到 $\mathcal{O}(n^{\log_2 7})$，即 $\omega = \log_2 7 \approx 2.8074$[2]，采用 Coppersmith–Winograd 矩阵乘法可以优化到 $\omega \leq 2.3755$[3]. 但这类小数点后的“用力过度”不是我们的菜，有时候反倒是滥用主定理，即它们所需的天文数字规模 $N \times N$ 的矩阵来临时，我们早该另觅出路，比如考虑使用 Jacobi 等迭代法。

公元二〇二四年九月二十四日，午时三刻，于 *HGX106* 室，惊闻徐夫子欲改弦更张，悲哉！

1 题目 1：*LU* 分解法的时间复杂度分析

1.1 题目描述

Prove that the time complexity of $\boldsymbol{LU}$ decomposition algorithm is $\mathcal{O}(n^3)$.

1.2 证明

$\boldsymbol{LU}$ 分解法的第一步是将系数矩阵 $\boldsymbol{A}$ 分解为一个下三角矩阵 $\boldsymbol{L}$ 和一个上三角矩阵 $\boldsymbol{U}$：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}.$$

[1]详见 *Numerical Recipes* §2.4

[2]有个直观而有趣的讨论，详见 *Numerical Recipes* §2.11

[3]$\omega < 2.404$ 的一种证明，参见 *MIT6.890 §23*

这一步常采用 Crout 方法实现，即在每一轮中，我们先计算 $\boldsymbol{L}$ 的第 k 列元素 l_{ik}，

$$l_{ik} = a_{ik} - \sum_{s=1}^{k-1} l_{is} u_{sk}, \qquad i = k, k+1, \ldots, n.$$

每一个 l_{ik} 的计算涉及 $k-1$ 次乘法和 $k-1$ 次减法，共有 $(n-k+1)$ 个 l_{ik} 需要计算；再计算 $\boldsymbol{U}$ 的第 k 行元素 u_{kj}，

$$u_{kj} = \frac{1}{l_{kk}} \left(a_{kj} - \sum_{s=1}^{k-1} l_{ks} u_{sj} \right), \qquad j = k+1, k+2, \ldots, n.$$

相比 l_{ik} 的计算多了一次除法，共有 $(n-k)$ 个 u_{kj} 需要计算，故第 k 轮的操作数为

$$(n-k+1) \cdot (2k-2) + (n-k) \cdot (2k-1) = -4k^2 + (4n+5)k - 3n - 2.$$

因此，分解步骤的总操作数为

$$T_c(n) = \sum_{k=1}^{n} \left[-4k^2 + (4n+5)k - 3n - 2 \right] = -4 \cdot \frac{n(n+1)(2n+1)}{6} + (4n+5) \cdot \frac{n(n+1)}{2} - (3n+2) \cdot n = \frac{4n^3 - 3n^2 - n}{6}.$$

再考虑回代步骤的操作数，即用分解得到的 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 求解方程组 $\boldsymbol{Ax} = \boldsymbol{b}$。首先求解 $\boldsymbol{Ly} = \boldsymbol{b}$，即

$$\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

这实质上是从第一行开始的 *Forward Substitution*，即

$$y_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} y_j \right).$$

每一步有 1 次除法，$(i-1)$ 次乘法与 $(i-1)$ 次减法；再求解 $\boldsymbol{Ux} = \boldsymbol{y}$，即

$$\begin{bmatrix} 1 & u_{12} & \cdots & u_{1n} \\ 0 & 1 & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},$$

这实质上是从最后一行开始的 *Backward Substitution*，即

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right).$$

每一步有 $(n-i)$ 次乘法与 $(n-i)$ 次减法，故回代步骤操作数为

$$T_s(n) = \sum_{i=1}^{n} \left[(2i-1) + (2n-2i) \right] = \sum_{i=1}^{n} (2n-1) = n(2n-1) = 2n^2 - n.$$

因此，$\boldsymbol{LU}$ 分解法的总操作数为

$$T(n) = T_c(n) + T_s(n) = \frac{4n^3 - 3n^2 - n}{6} + 2n^2 - n = \frac{4n^3 + 9n^2 - 7n}{6}.$$

其中有除法 $n(n+1)/2$ 次，乘法与减法各 $n(n-1)(2n+5)/6$ 次，故

$$\boxed{T(n) = \mathcal{O}(n^3)}$$

Amazing，*居然与 Gaussian 消元法的各种操作数都相同！*

2 题目 2：结合部分主元应用高斯消元法

2.1 题目描述

Using **partial pivoting** Gaussian elimination to solve the system of equations:

$$\begin{cases} 2x_1 + 3x_2 + 5x_3 = 5 \\ 3x_1 + 4x_2 + 8x_3 = 6 \\ x_1 + 3x_2 + 3x_3 = 5 \end{cases}$$

2.2 程序描述

本题要求结合部分主元法，也就是每次要从当前列中选取绝对值最大的元素作为主元，提升数值稳定性。具体思路如第 1 节所述，只是在必需时加上交换行的操作。虽然题目要求解的方程组具有唯一解

$$x_1 = 2, \quad x_2 = 2, \quad x_3 = -1,$$

但是为了保证程序的通用性，我们仍然考虑了可能出现的无穷多组解、无解的情况，这借助于 `methods.cpp` 中的 `DetermineRank` 来计算矩阵的秩，与 `CheckConsistency` 来检查（行阶梯化之后的）增广矩阵是否无解。当被判定为非列满秩（即秩小于系数矩阵列数）时，我们将调用 `ShowGeneralSolution` 来输出通解，否则正常执行回代法输出唯一解。本题子目录结构如下

```
|-- doxygen_output
|   |-- html
|   `-- latex
|-- problem_2.tex
|-- readme.html
`-- src
    |-- Gaussian.exe
    |-- interaction.cpp
    |-- interaction.h
    |-- main.cpp
    |-- methods.cpp
    |-- methods.h
    |-- utils.cpp
    |-- utils.h
    |-- quiz.in
    |-- inf.in
    |-- inf_2.in
    |-- no.in
    |-- pi_27.in
    `-- pi_81.in
```

助教老师审阅源代码时，可借助 `readme.html` 便捷查看 Doxygen 生成的注释文档。在 `src` 目录下，运行`g++ *.cpp -o main`（或其它编译器，需要支持`-std=c++11` 标准）编译，再在当前目录使用`./main` 运行即可（也有已经编译好的 `Gaussian.exe`，适配 Win64）。`interaction.cpp` 负责交互功能，包括在当前文件夹搜索`.in` 文件供用户选择等；`main.cpp` 是主程序入口点，其逻辑结构在伪代码 1 中有详细说明；`methods.cpp` 负责算法实现，包括使用高斯消元法行阶梯化、计算秩、检查方程组自洽性、回代法求唯一解等，逻辑结构在伪代码 2,3,4,5 中有详细说明；`utils.cpp` 包含一些通用的工具函数，如 `ReadMatrix`,`ShowMatrix` 等，并提供计时功能。目录下还准备了 6 个测试用的`.in` 文件，其中 `quiz.in` 是本题要求的输入文件，`inf.in` 是约束重复导致无穷多组解的例子，`inf_2.in` 是方程少于未知数的例子，`no.in` 是无解的例子，`pi_27.in` 和 `pi_81.in`，分别是从圆周率生成的 27×28 和 81×82 的增广矩阵，用于验证前述算法时间复杂度的分析，最终结果表明，两者运行时间之比为 $3.43s : 62.7s \approx 1 : 18$，考虑到输入输出等影响，近似吻合 $O(n^3) = 27$ 的时间复杂度之比。同时还借助 `numpy` 库的 `linalg` 模块在服务器上求解了 `pi_81.in`，其结果与本程序输出一致（还快些），验证了本算法的正确性，详细的结果分析见2.4所述。

2.3 伪代码

Algorithm 1: Gaussian Elimination Solver

```
Input: Augmented Matrix (float,shape=(m,n)) ;              // The augmented matrix from .in file
Output: Solutions (array) ;                      // May be no solution or parameterized solution
while True do
    selected_file ← SelectInputFile() ;                                 // Select the input file
    if selected_file is empty then
        exit ;                                                    // Exit if no file is selected
    end
    InitMatrix(matrix, rows, cols, selected_file) ;                     // Initialize the matrix
    ShowEquations(matrix, rows, cols) ;                       // Display the system of equations
    exchange_count ← GaussianElimination(matrix, rows, cols) ;   // Perform Gaussian elimination
     and record row exchanges
    rank ← DetermineRank(matrix, rows, cols) ;               // Determine the rank of the matrix
    consistent ← CheckConsistency(matrix, rows, cols) ;     // Check if the system is consistent
    if not consistent then
        DisplaySolution("No solution") ;                          // Display no solution message
    end
    else if rank < (cols − 1) then
        DisplaySolution("Parameterized solution") ;          // Display parameterized solution
    end
    else
        solution ← BackSubstitution(matrix, rows, cols) ;           // Perform back substitution
        if solution exists then
            DisplaySolution(solution) ;                           // Display the unique solution
        end
        else
            DisplaySolution("No solution") ;    // If no solution exists, display no solution
        end
    end
    choice ← AskRunAgain() ;                             // Ask if the user wants to run again
    if choice ≠ 'y' and choice ≠ 'Y' then
        break ;                               // Exit loop if the choice is not 'y' or 'Y'
    end
end
WaitForExit() ;                                                     // Wait for program exit
```

Algorithm 2: Gaussian Elimination with Partial Pivoting

Input: `matrix` (Matrix), `rows` (int), `cols` (int)

Output: `exchange_count` (int)

```
exchange_count ← 0;
for k ← 0 to cols − 2 do
    pivot ← PartialPivoting(matrix, k, rows) ;                    // Select pivot row
    if pivot ≠ k then
        SwapRows(matrix, k, pivot) ;                              // Swap rows for pivoting
        exchange_count ← exchange_count + 1;
    end
    for i ← k + 1 to rows − 1 do
        factor ← matrix[i][k] / matrix[k][k] ;                    // Compute elimination factor
        for j ← k to cols − 1 do
            matrix[i][j] ← matrix[i][j] - factor · matrix[k][j] ; // Update matrix entry
        end
    end
end
return exchange_count;
```

Algorithm 3: Determine Rank

Input: `matrix` (Matrix), `rows` (int), `cols` (int)

Output: `rank` (int)

```
rank ← 0;
for i ← 1 to rows do
    for j ← 1 to cols − 1 do
        if matrix[i][j] ≠ 0 then
            rank ← rank + 1 ;       // Check non-zero element in row except last column
            break;
        end
    end
end
return rank;
```

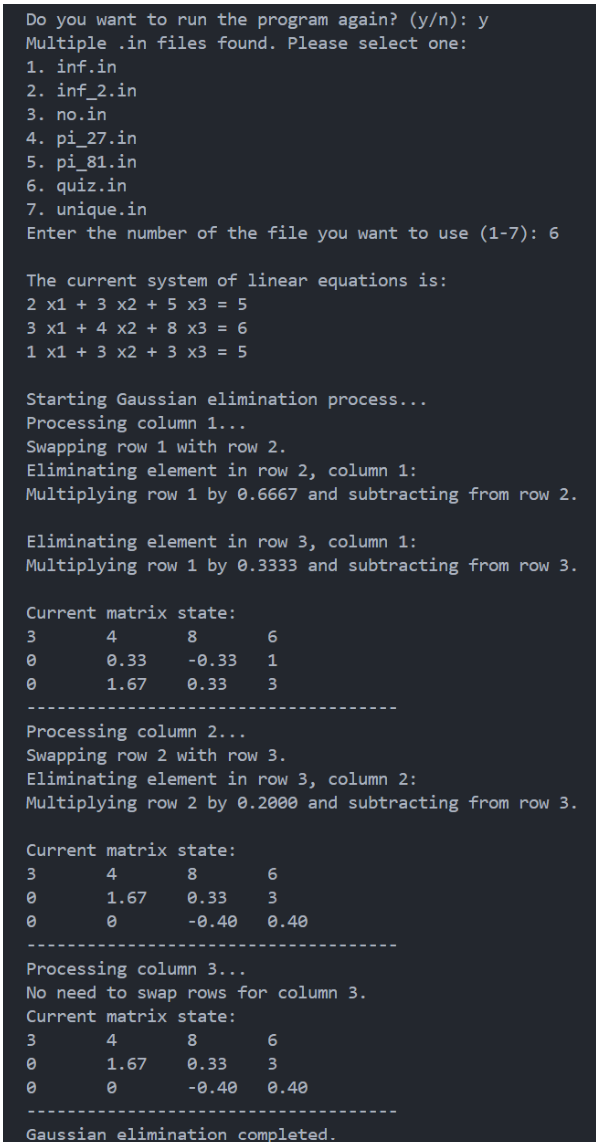
Algorithm 4: Check Consistency

```
Input: matrix (Matrix), rows (int), cols (int)
Output: consistent (bool)
for i ← 0 to rows − 1 do
    all_zero ← true;
    for j ← 0 to cols − 2 do
        if matrix[i][j] ≠ 0 then
            all_zero ← false;
            break;
        end
    end
    if all_zero and matrix[i][cols − 1] ≠ 0 then
        return false ;                    // Inconsistent equation detected
    end
end
return true;
```

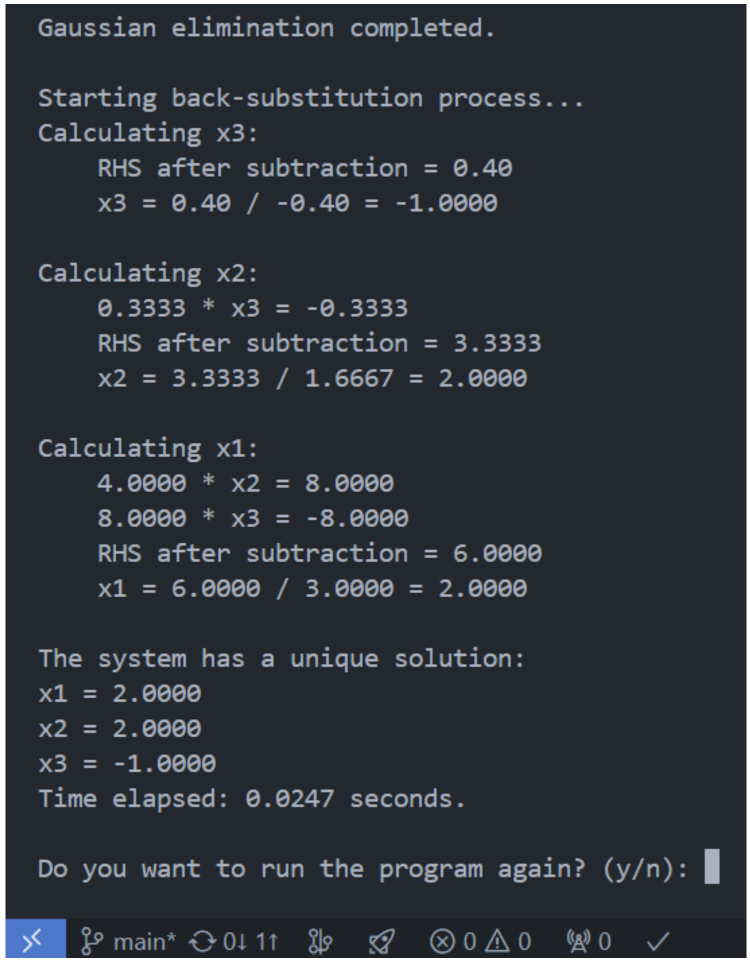
Algorithm 5: Back Substitution

```
Input: matrix (Matrix), rows (int), cols (int)
Output: solution (Vector)
solution ← Vector(cols - 1);
for i ← rows − 1 downto 0 do
    sum ← 0;
    for j ← i + 1 to cols − 2 do
        sum ← sum + (matrix[i][j] · solution[j]);
    end
    if matrix[i][i] == 0 then
        return solution does not exist ;    // Division by zero implies no unique solution
    end
    solution[i] ← (matrix[i][cols − 1] − sum)/matrix[i][i] ;    // Compute solution for variable i
end
return solution;
```

2.4 结果示例



```
Do you want to run the program again? (y/n): y
Multiple .in files found. Please select one:
1. inf.in
2. inf_2.in
3. no.in
4. pi_27.in
5. pi_81.in
6. quiz.in
7. unique.in
Enter the number of the file you want to use (1-7): 6

The current system of linear equations is:
2 x1 + 3 x2 + 5 x3 = 5
3 x1 + 4 x2 + 8 x3 = 6
1 x1 + 3 x2 + 3 x3 = 5

Starting Gaussian elimination process...
Processing column 1...
Swapping row 1 with row 2.
Eliminating element in row 2, column 1:
Multiplying row 1 by 0.6667 and subtracting from row 2.

Eliminating element in row 3, column 1:
Multiplying row 1 by 0.3333 and subtracting from row 3.

Current matrix state:
3       4       8       6
0       0.33    -0.33   1
0       1.67    0.33    3
-----------------------------------
Processing column 2...
Swapping row 2 with row 3.
Eliminating element in row 3, column 2:
Multiplying row 2 by 0.2000 and subtracting from row 3.

Current matrix state:
3       4       8       6
0       1.67    0.33    3
0       0       -0.40   0.40
-----------------------------------
Processing column 3...
No need to swap rows for column 3.
Current matrix state:
3       4       8       6
0       1.67    0.33    3
0       0       -0.40   0.40
-----------------------------------
Gaussian elimination completed.
```



```
Gaussian elimination completed.

Starting back-substitution process...
Calculating x3:
    RHS after subtraction = 0.40
    x3 = 0.40 / -0.40 = -1.0000

Calculating x2:
    0.3333 * x3 = -0.3333
    RHS after subtraction = 3.3333
    x2 = 3.3333 / 1.6667 = 2.0000

Calculating x1:
    4.0000 * x2 = 8.0000
    8.0000 * x3 = -8.0000
    RHS after subtraction = 6.0000
    x1 = 6.0000 / 3.0000 = 2.0000

The system has a unique solution:
x1 = 2.0000
x2 = 2.0000
x3 = -1.0000
Time elapsed: 0.0247 seconds.

Do you want to run the program again? (y/n):
```


图 1: 原题要求解的 `quiz.in`

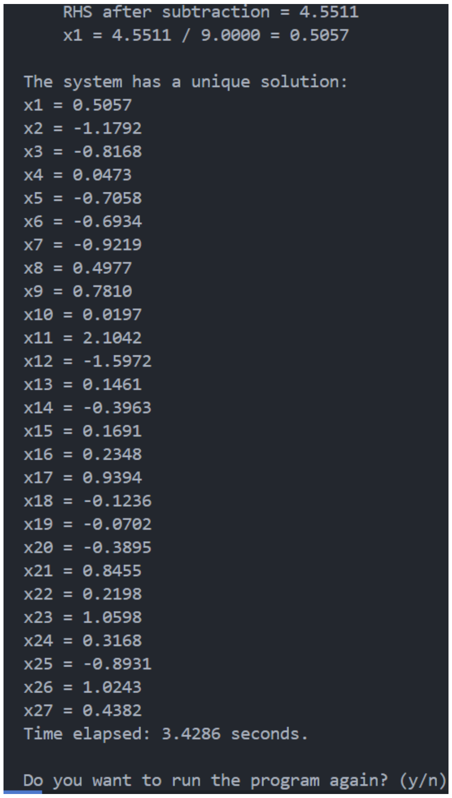
```
The current system of linear equations is:
1 x1 + 2 x2 + 3 x3 = 4
2 x1 + 4 x2 + 6 x3 = 8
1 x1 + 2 x2 + 3 x3 = 5

Starting Gaussian elimination process...
Processing column 1...
Swapping row 1 with row 2.
Eliminating element in row 2, column 1:
Multiplying row 1 by 0.5000 and subtracting from row 2.

Eliminating element in row 3, column 1:
Multiplying row 1 by 0.5000 and subtracting from row 3.

Current matrix state:
2       4       6       8
0       0       0       0
0       0       0       1
-------------------------------------
Processing column 2...
No need to swap rows for column 2.
Warning: Pivot element in row 2 is close to zero. The matrix may be singular.
Processing column 3...
No need to swap rows for column 3.
Warning: Pivot element in row 3 is close to zero. The matrix may be singular.
Gaussian elimination completed.

The system of equations is inconsistent and has no solution.
Time elapsed: 0.0094 seconds.
```

图 2: 无解情形 `no.in`

```
The current system of linear equations is:
1 x1 + 2 x2 + 3 x3 = 6
2 x1 + 4 x2 + 6 x3 = 12
3 x1 + 6 x2 + 9 x3 = 18

Starting Gaussian elimination process...
Processing column 1...
Swapping row 1 with row 3.
Eliminating element in row 2, column 1:
Multiplying row 1 by 0.6667 and subtracting from row 2.

Eliminating element in row 3, column 1:
Multiplying row 1 by 0.3333 and subtracting from row 3.

Current matrix state:
3       6       9       18
0       0       0       0
0       0       0       0
-------------------------------------
Processing column 2...
No need to swap rows for column 2.
Warning: Pivot element in row 2 is close to zero. The matrix may be singular.
Processing column 3...
No need to swap rows for column 3.
Warning: Pivot element in row 3 is close to zero. The matrix may be singular.
Gaussian elimination completed.

The system has infinitely many solutions.
Solution space dimension: 2
General solution:
x = [6.0000, 0.0000, 0.0000] + t1 * [-2.0000, 1.0000, 0.0000] +  + t2 * [-3.0000, 0.0000, 1.0000]

Time elapsed: 0.0135 seconds.
```

```
Do you want to run the program again? (y/n): y
Multiple .in files found. Please select one:
1. inf.in
2. inf_2.in
3. no.in
4. pi_27.in
5. pi_81.in
6. quiz.in
Enter the number of the file you want to use (1-6): 2

The current system of linear equations is:
1 x1 + 2 x2 + 3 x3 + 4 x4 = 5
6 x1 + 7 x2 + 8 x3 + 9 x4 = 10

Starting Gaussian elimination process...
Processing column 1...
Swapping row 1 with row 2.
Eliminating element in row 2, column 1:
Multiplying row 1 by 0.1667 and subtracting from row 2.

Current matrix state:
6       7       8       9       10
0       0.83    1.67    2.50    3.33
-------------------------------------
Processing column 2...
No need to swap rows for column 2.
Current matrix state:
6       7       8       9       10
0       0.83    1.67    2.50    3.33
-------------------------------------
Gaussian elimination completed.

The system has infinitely many solutions.
Solution space dimension: 2
General solution:
x = [-3.0000, 4.0000, 0.0000, 0.0000] + t1 * [1.0000, -2.0000, 1.0000, 0.0000] +  + t2 * [2.0000, -3.0000, 0.0000, 1.0000]

Time elapsed: 0.0111 seconds.
```

图 3: 两种无穷多组解情形 `inf.in`,`inf_2.in`



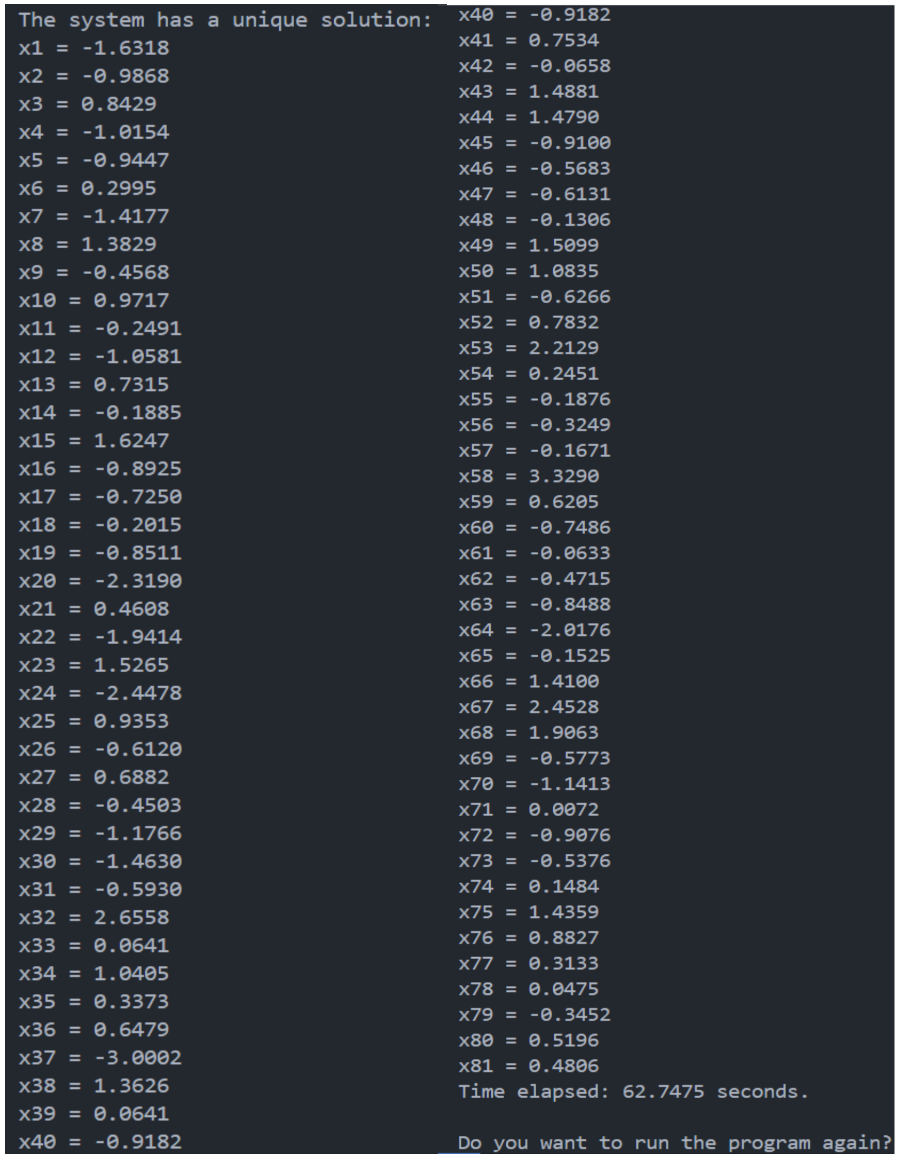


图 4: 圆周率提取的 `pi_27.in` 和 `pi_81.in` 对比

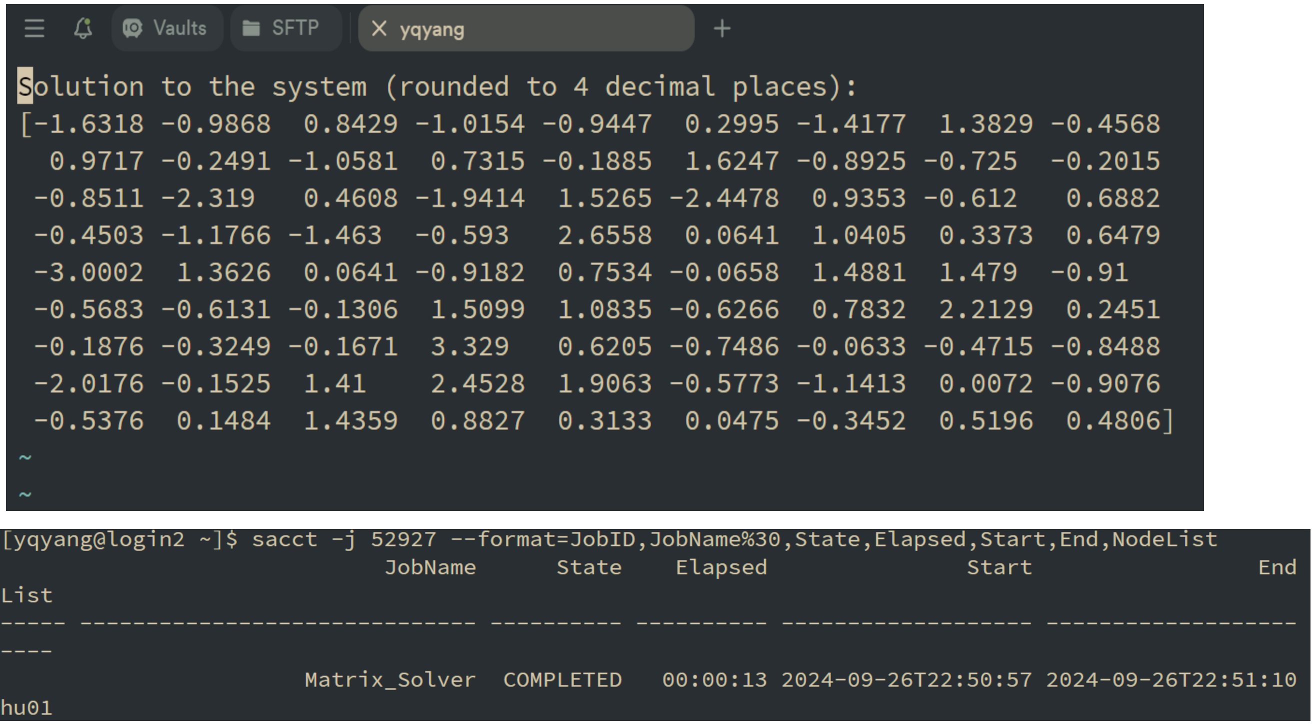


图 5: `pi_81.in` 使用 `numpy` 库求解的结果

3 题目 3：变分法求解一维薛定谔方程

3.1 题目描述

Solve the 1D Schrödinger 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)

$$\phi_i(x) = (\frac{\nu_i}{\pi})^{1/2} \mathrm{e}^{-\nu_i (x-s_i)^2}.$$

Consider the three lowest energy eigenstates.

3.2 程序描述

3.3 伪代码

3.4 结果示例