计算物理作业 3

杨远青　22300190015　CompPhys 24

2024 年 10 月 12 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 ***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 ***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 ***LU*** decomposition algorithm is $\mathcal{O}(n^3)$.

1.2 证明

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
|-- Doxygen.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 `Doxygen.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 $k \leftarrow 0$ to $cols - 2$ do
    pivot ← PartialPivoting(matrix, $k$, rows) ;                    // Select pivot row
    if $pivot \neq k$ then
        SwapRows(matrix, $k$, pivot) ;                              // Swap rows for pivoting
        exchange_count ← exchange_count + 1;
    end
    for $i \leftarrow k + 1$ to $rows - 1$ do
        factor ← matrix[$i$][$k$] / matrix[$k$][$k$] ;              // Compute elimination factor
        for $j \leftarrow k$ 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 $i \leftarrow 1$ to $rows$ do
    for $j \leftarrow 1$ to $cols - 1$ do
        if $matrix[i][j] \neq 0$ then
            rank ← rank + 1 ;          // Check non-zero element in row except last column
            break;
        end
    end
end
return rank;
```

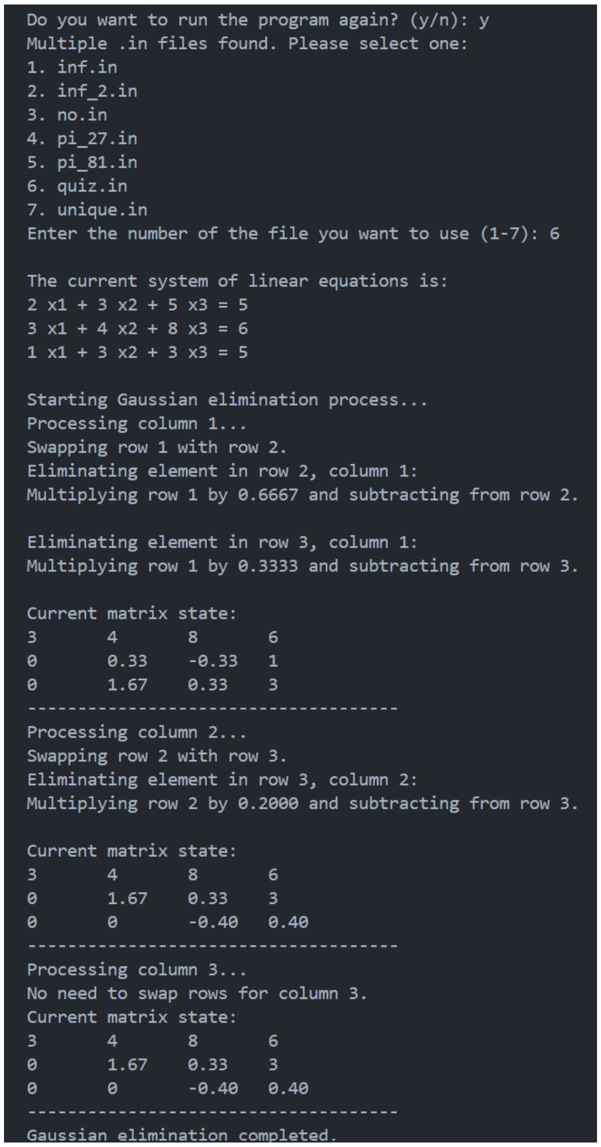
Algorithm 4: Check Consistency

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

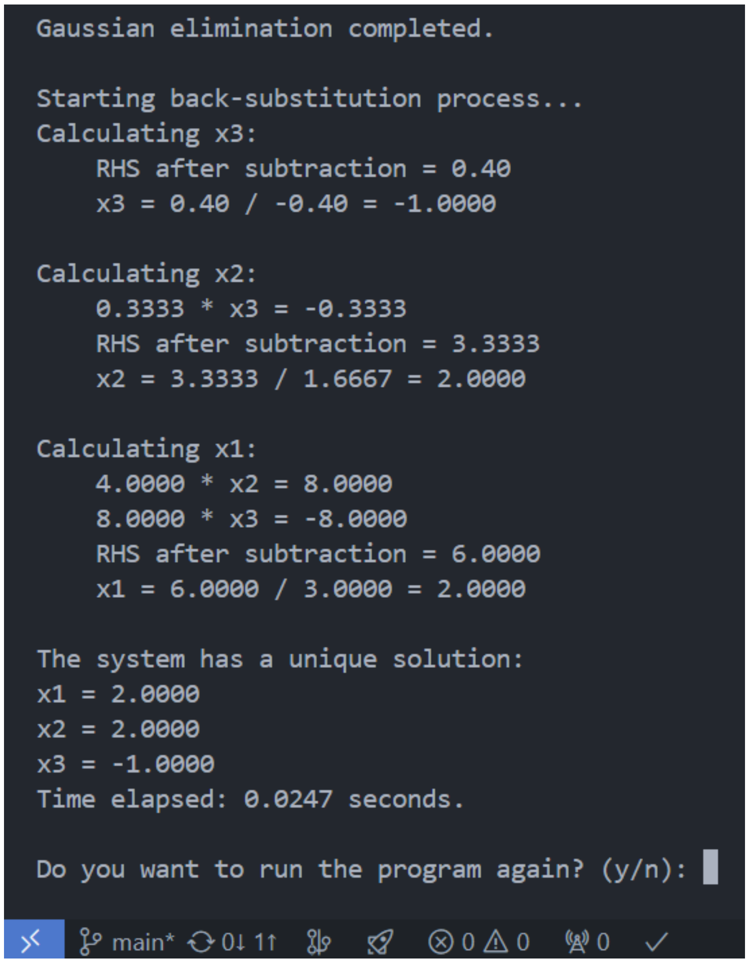
Algorithm 5: Back Substitution

```
Input: matrix (Matrix), rows (int), cols (int)
Output: solution (Vector)
solution ← Vector(cols - 1);
for $i \leftarrow rows - 1$ downto 0 do
    sum ← 0;
    for $j \leftarrow i + 1$ 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

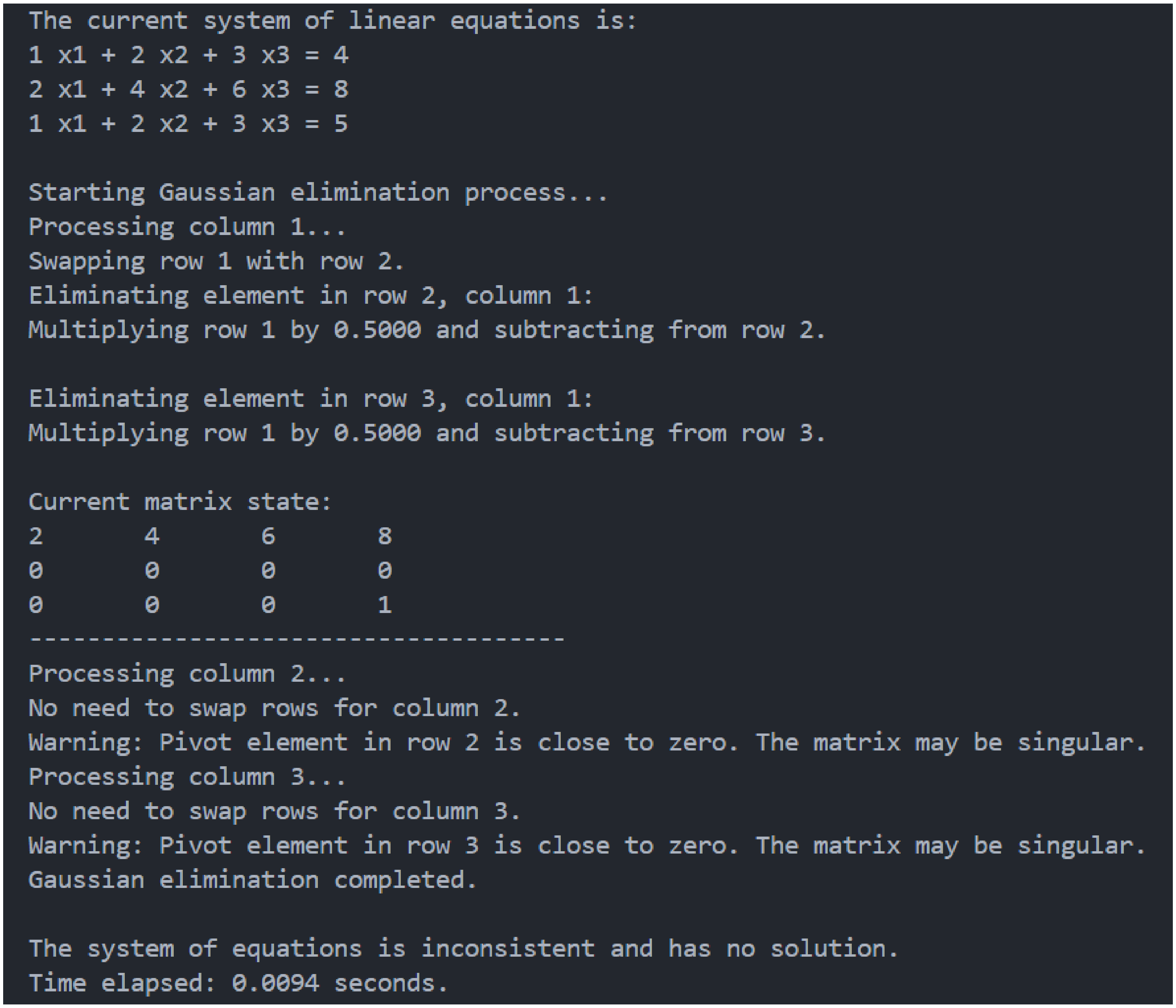


```
Do you want to run the program again? (y/n): y
Multiple .in files found. Please select one:
1. inf.in
2. inf_2.in
3. no.in
4. pi_27.in
5. pi_81.in
6. quiz.in
7. unique.in
Enter the number of the file you want to use (1-7): 6

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.
```



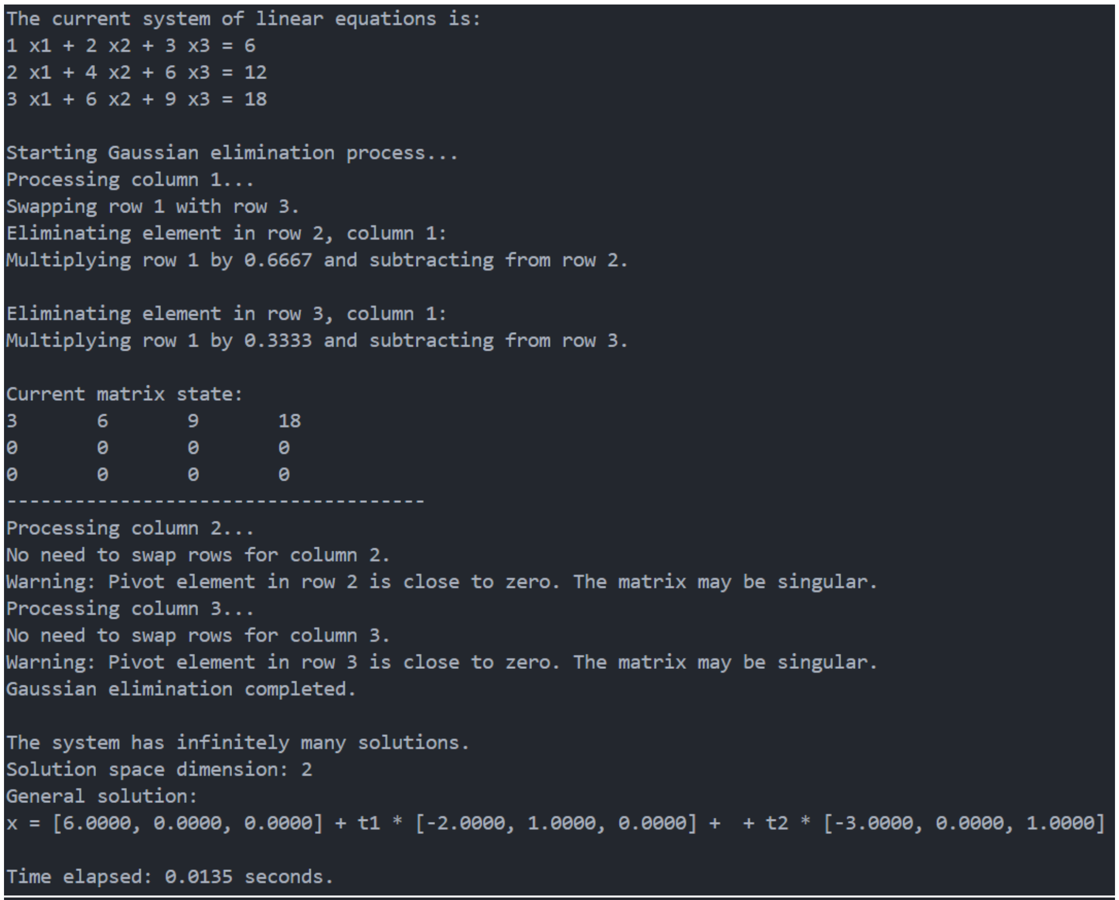
```
Gaussian elimination completed.

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

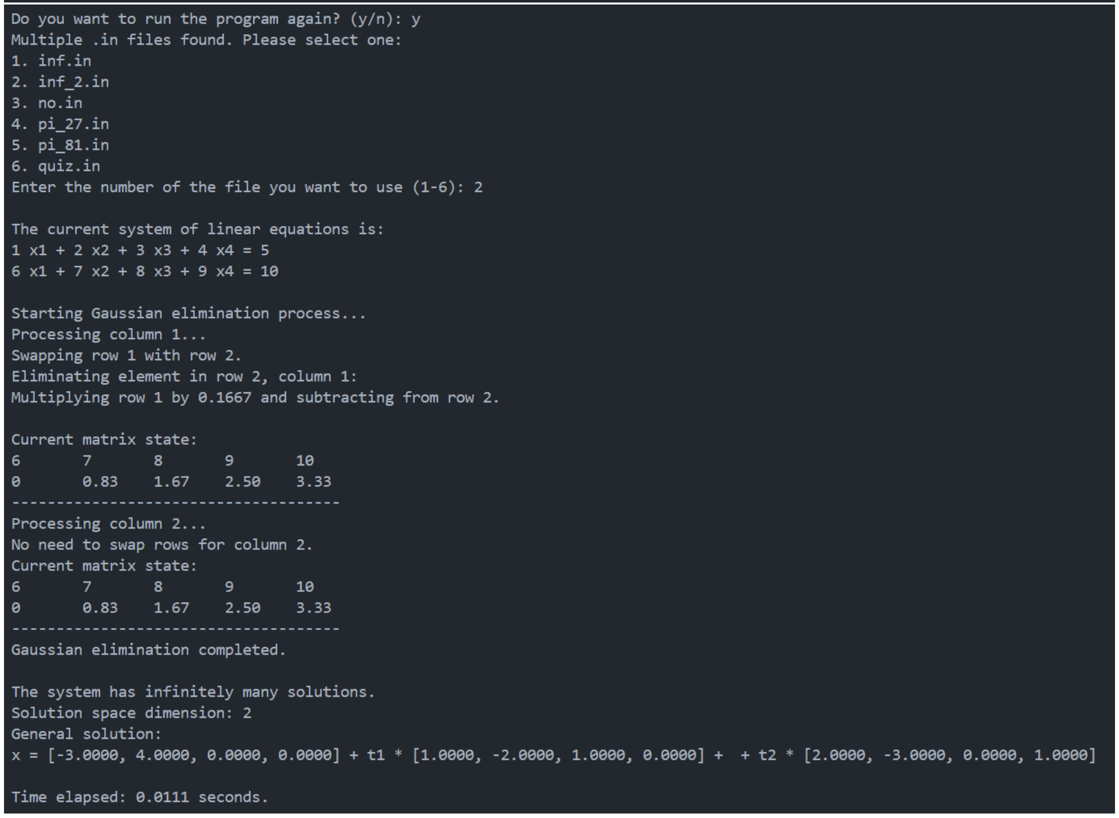


```
The current system of linear equations is:
1 x1 + 2 x2 + 3 x3 = 4
2 x1 + 4 x2 + 6 x3 = 8
1 x1 + 2 x2 + 3 x3 = 5

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`

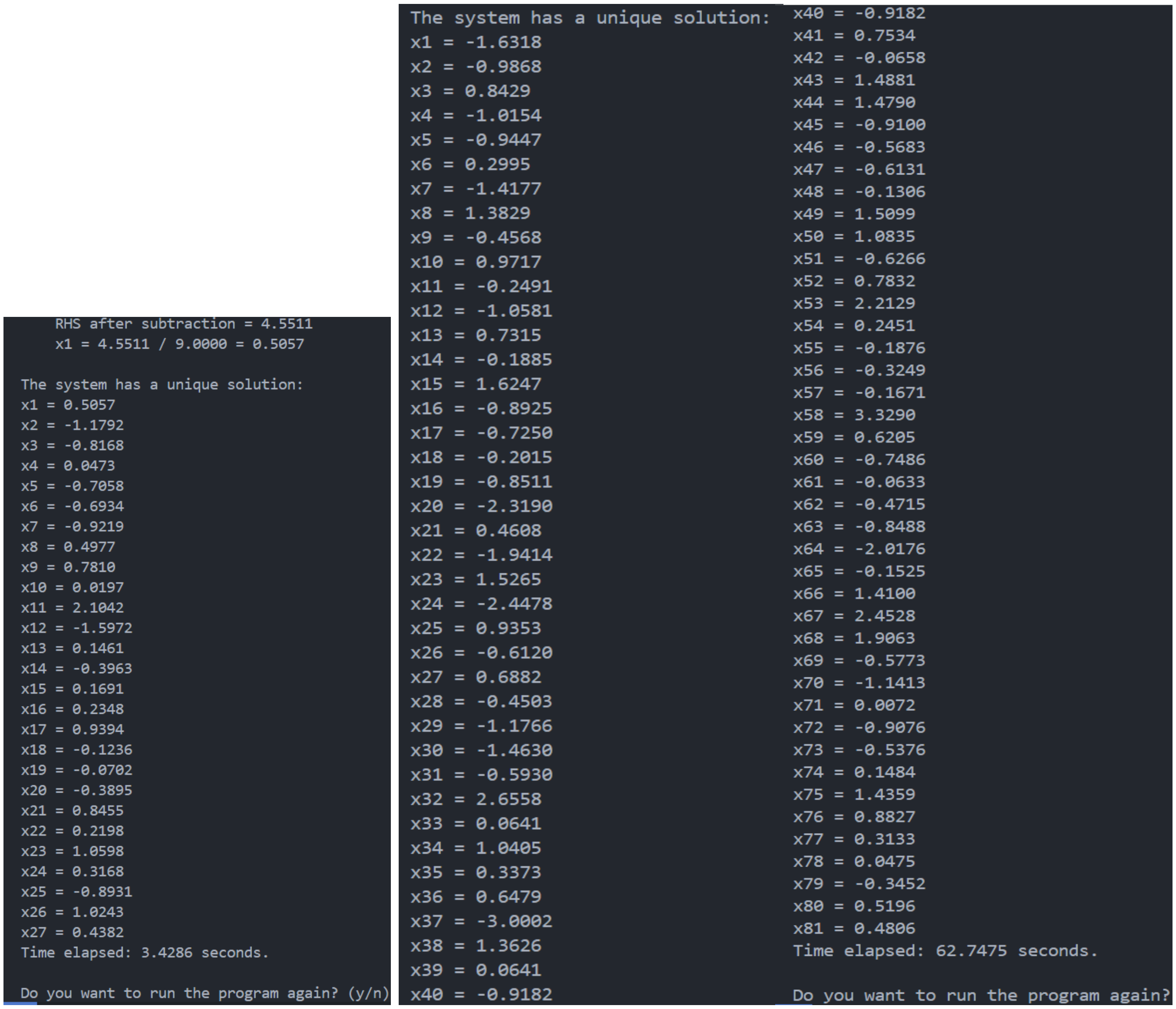


```
The current system of linear equations is:
1 x1 + 2 x2 + 3 x3 = 6
2 x1 + 4 x2 + 6 x3 = 12
3 x1 + 6 x2 + 9 x3 = 18

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`

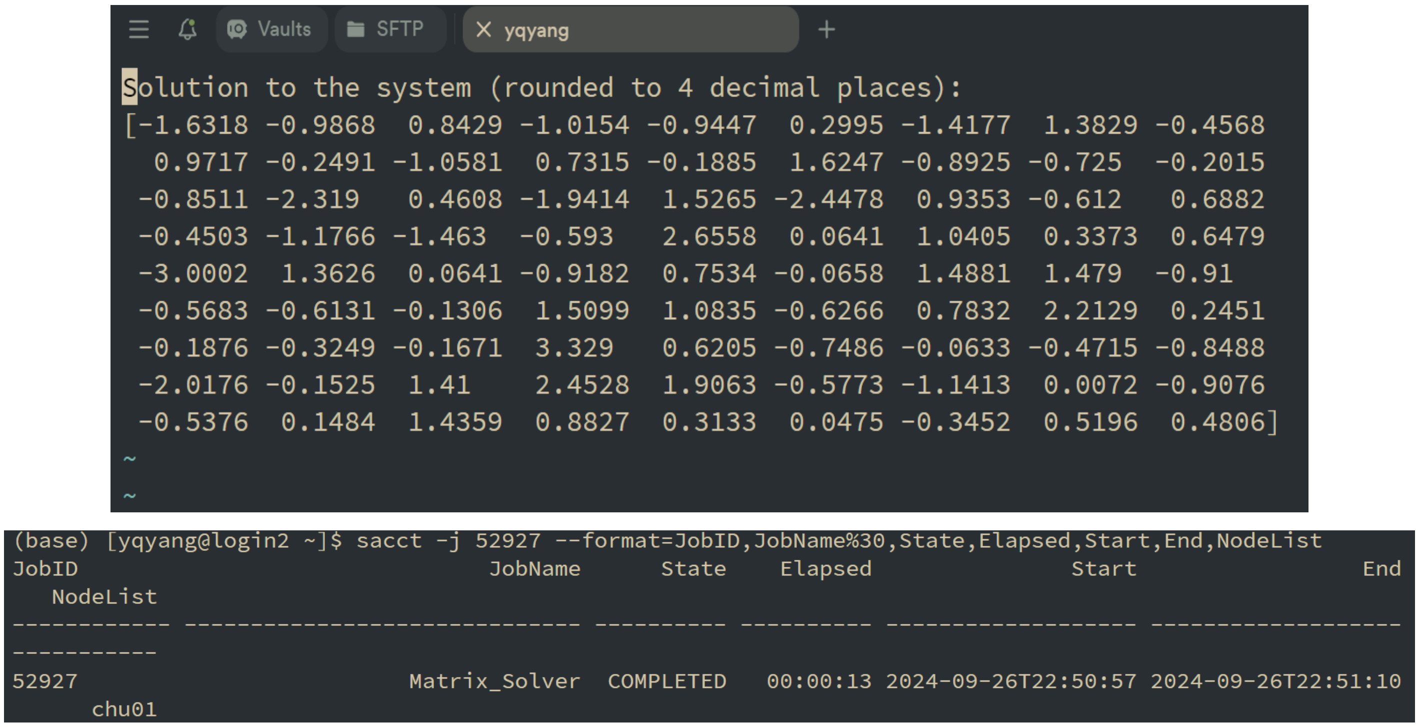


```
    RHS after subtraction = 4.5511
    x1 = 4.5511 / 9.0000 = 0.5057

The system has a unique solution:
x1 = 0.5057
x2 = -1.1792
x3 = -0.8168
x4 = 0.0473
x5 = -0.7058
x6 = -0.6934
x7 = -0.9219
x8 = 0.4977
x9 = 0.7810
x10 = 0.0197
x11 = 2.1042
x12 = -1.5972
x13 = 0.1461
x14 = -0.3963
x15 = 0.1691
x16 = 0.2348
x17 = 0.9394
x18 = -0.1236
x19 = -0.0702
x20 = -0.3895
x21 = 0.8455
x22 = 0.2198
x23 = 1.0598
x24 = 0.3168
x25 = -0.8931
x26 = 1.0243
x27 = 0.4382
Time elapsed: 3.4286 seconds.

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

```
The system has a unique solution:
x1 = -1.6318
x2 = -0.9868
x3 = 0.8429
x4 = -1.0154
x5 = -0.9447
x6 = 0.2995
x7 = -1.4177
x8 = 1.3829
x9 = -0.4568
x10 = 0.9717
x11 = -0.2491
x12 = -1.0581
x13 = 0.7315
x14 = -0.1885
x15 = 1.6247
x16 = -0.8925
x17 = -0.7250
x18 = -0.2015
x19 = -0.8511
x20 = -2.3190
x21 = 0.4608
x22 = -1.9414
x23 = 1.5265
x24 = -2.4478
x25 = 0.9353
x26 = -0.6120
x27 = 0.6882
x28 = -0.4503
x29 = -1.1766
x30 = -1.4630
x31 = -0.5930
x32 = 2.6558
x33 = 0.0641
x34 = 1.0405
x35 = 0.3373
x36 = 0.6479
x37 = -3.0002
x38 = 1.3626
x39 = 0.0641
x40 = -0.9182
```

```
x40 = -0.9182
x41 = 0.7534
x42 = -0.0658
x43 = 1.4881
x44 = 1.4790
x45 = -0.9100
x46 = -0.5683
x47 = -0.6131
x48 = -0.1306
x49 = 1.5099
x50 = 1.0835
x51 = -0.6266
x52 = 0.7832
x53 = 2.2129
x54 = 0.2451
x55 = -0.1876
x56 = -0.3249
x57 = -0.1671
x58 = 3.3290
x59 = 0.6205
x60 = -0.7486
x61 = -0.0633
x62 = -0.4715
x63 = -0.8488
x64 = -2.0176
x65 = -0.1525
x66 = 1.4100
x67 = 2.4528
x68 = 1.9063
x69 = -0.5773
x70 = -1.1413
x71 = 0.0072
x72 = -0.9076
x73 = -0.5376
x74 = 0.1484
x75 = 1.4359
x76 = 0.8827
x77 = 0.3133
x78 = 0.0475
x79 = -0.3452
x80 = 0.5196
x81 = 0.4806
Time elapsed: 62.7475 seconds.

Do you want to run the program again?
```

图 4: 圆周率提取的 pi_27.in 和 pi_81.in 对比



```
Solution to the system (rounded to 4 decimal places):
[-1.6318 -0.9868  0.8429 -1.0154 -0.9447  0.2995 -1.4177  1.3829 -0.4568
  0.9717 -0.2491 -1.0581  0.7315 -0.1885  1.6247 -0.8925 -0.725  -0.2015
 -0.8511 -2.319   0.4608 -1.9414  1.5265 -2.4478  0.9353 -0.612   0.6882
 -0.4503 -1.1766 -1.463  -0.593   2.6558  0.0641  1.0405  0.3373  0.6479
 -3.0002  1.3626  0.0641 -0.9182  0.7534 -0.0658  1.4881  1.479  -0.91
 -0.5683 -0.6131 -0.1306  1.5099  1.0835 -0.6266  0.7832  2.2129  0.2451
 -0.1876 -0.3249 -0.1671  3.329   0.6205 -0.7486 -0.0633 -0.4715 -0.8488
 -2.0176 -0.1525  1.41    2.4528  1.9063 -0.5773 -1.1413  0.0072 -0.9076
 -0.5376  0.1484  1.4359  0.8827  0.3133  0.0475 -0.3452  0.5196  0.4806]
~
~
```

```
(base) [yqyang@login2 ~]$ sacct -j 52927 --format=JobID,JobName%30,State,Elapsed,Start,End,NodeList
JobID                               JobName      State    Elapsed               Start                 End
   NodeList
------------ ------------------------------ ---------- ---------- ------------------- -------------------
-----------
52927                         Matrix_Solver  COMPLETED   00:00:13 2024-09-26T22:50:57 2024-09-26T22:51:10
      chu01
```

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

长海今天人好多啊

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)

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

Consider the three lowest energy eigenstates.

3.2 程序描述

本题要求用原版高斯基线性组合，通过变分原理找能级。高斯基并不相互正交，故需要计算重叠积分 S_{ij}：

$$S_{ij} = \left(\frac{v_i v_j}{v_i + v_j}\right)^{1/2} \exp\left(-\frac{v_i v_j (s_i - s_j)^2}{v_i + v_j}\right),$$

接着计算动能积分 T_{ij}（此解答采用自然单位制 $\hbar = m = 1$）：

$$T_{ij} = \left(\frac{v_i^{3/2} v_j^{3/2}}{\sqrt{\pi}(v_i + v_j)^{5/2}}\right) \left[(v_i + v_j) - 2v_i v_j (s_i - s_j)^2\right] \exp\left(-\frac{v_i v_j (s_i - s_j)^2}{v_i + v_j}\right).$$

而对于势能积分 V_{ij}，通过广义本征值问题 $\mathbf{Hc} = E\mathbf{Sc}$ 可以找到能级 E 和对应的波函数 $\psi = \mathbf{c}\phi$。计算的难点在于，对于一般的势能 $V(x)$，高斯基下的积分

$$V_{ij} = \int \phi_i(x)V(x)\phi_j(x)\,dx$$

没有解析解，需要数值积分来求解。幸运的是，本题所求的势能 $V(x)$ 是多项式形式，使用 Mathematica® 可以方便地求解 x^n 对应的势能积分。然而，高阶积分的解析表达式会变得非常复杂。GPT 提醒我，两个高斯基的乘积仍是一个新的高斯函数，因此可利用该特性来简化多项式势能的积分，即通过高斯分布的矩来处理。两个高斯函数的乘积为

$$\phi_i(x)\phi_j(x) = \left(\frac{2v_iv_j}{\pi^2}\right)^{1/4} \exp\left(-v_i(x-s_i)^2 - v_j(x-s_j)^2\right).$$

将 $v_i(x-s_i)^2 + v_j(x-s_j)^2$ 展开为：

$$(v_i+v_j)\left(x - \frac{v_is_i+v_js_j}{v_i+v_j}\right)^2 + \frac{v_iv_j(s_i-s_j)^2}{v_i+v_j}.$$

因此，$\phi_i(x)\phi_j(x)$ 可以写成一个新的高斯函数的形式：

$$\phi_i(x)\phi_j(x) = \left(\frac{2v_iv_j}{\pi(v_i+v_j)}\right)^{1/2} \exp\left(-\frac{v_iv_j(s_i-s_j)^2}{v_i+v_j}\right) \exp\left(-(v_i+v_j)\,(x-\mu)^2\right),$$

其中新的高斯分布的中心 μ 和方差 σ^2 为：

$$\mu = \frac{v_is_i+v_js_j}{v_i+v_j}, \quad \sigma^2 = \frac{1}{2(v_i+v_j)}.$$

于是对于多项式形式的势能 $V(x) = x^n$，我们可以将势能积分转化为新高斯分布的矩问题。高斯分布 $G(x;\mu,\sigma^2)$ 的 n 阶矩 $M_n = \mathbb{E}[x^n]$ 满足递推关系

$$M_n(\mu,\sigma^2) = \mu M_{n-1}(\mu,\sigma^2) + (n-1)\sigma^2 M_{n-2}(\mu,\sigma^2),$$

其中

$$M_0 = 1, \quad M_1 = \mu.$$

因此，前几个矩的具体值为

$$M_2 = \mu^2 + \sigma^2,$$

$$M_3 = \mu^3 + 3\mu\sigma^2,$$

$$M_4 = \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4.$$

综上，势能积分可以表示为

$$V_{ij} = S_{ij} \cdot M_n(\mu,\sigma^2),$$

其中 S_{ij} 是重叠积分，保证新高斯分布的归一化，M_n 是高斯分布的 n 阶矩。通过递推公式可以更方便地计算多项式势能下的高斯积分，源代码中实际使用 `Switch` 语句处理了 $n \in [1,4]$ 的情形。最后的本征值求解借助了 *Julia* `LinearAlgebra` 库中的 `eigen` 函数，并采用梯形法计算最后波函数的归一化系数。

本题子目录结构如下

```
|-- Documenter.html
|-- figs
|-- problem_3.tex
`-- src
    |-- documenter_output
    |-- integrals.nb
    |-- interaction.jl
    |-- main.jl
    |-- methods.jl
    `-- utils.jl
```

助教老师审阅源代码时，可借助 `Documenter.html` 便捷查看 `Documenter` 生成的注释文档。在 `src` 目录下，运行`julia main.jl` 即可运行程序（需安装 `LinearAlgebra` 与 `Plots` 包），`main.jl` 是主程序入口点，其逻辑结构在伪代码 6 中有详细说明；`methods.jl` 负责算法实现，包括重叠积分，动能积分，势能积分与求解 Schrödinger 方程等，逻辑结构在伪代码 7,8 中有详细说明；`interaction.jl` 负责交互功能，包括展示主菜单，根据不同选项处理用户输入输出等；`utils.jl` 包含一些通用的工具函数，如梯形法，波函数归一化，异常处理等。目录下还准备了 `integrals.nb`，是用于验证高斯积分的 Mathematica® 笔记本文件，最终源代码采用前述的多级矩计算方法，比直接录入 Mathematica® 结果更加高效简洁且具备拓展性[4]。

程序主菜单提供了四个选项，分别对应使用：自定义参数求解题目要求的两种势阱、使用默认参数一次求解两种势阱，以及自定义四次以内的多项式势能进行求解。自定义参数包括高斯基数量、宽度 v、中心 s（在子模式下可选择变动其中之一，为另一个指定范围），输出能级数，自定义势阱系数等。用户还可以选择是否绘制波函数图像，但归一化处理较为耗时，故内置了多线程处理不同能级绘制，但依赖于用户授权，故建议至少使用`julia -t 4 main.jl` 运行，可显著提升绘图速度。输出结果与绘图详见3.4节所示。

碎碎念：我多次尝试使用 `PackageCompiler` 打包编译本程序，奈何依赖屡屡出错，也没有合适的压缩方法，更不用谈交叉编译；尝试过使用 `Pluto` 交互式改造，发布至 Binder，但新语法过多，中道崩殂。尽管我非常欣赏 Julia 超前的各项设计理念，风格优美在一众科学计算语言中当属清流，但生态实在还是不够配套，继续加油吧！

[4]猛然想起钟阳老师说 DeepMD® 就是因为内置了许多积分查找的优化，速度提升不少。

3.3 伪代码

Algorithm 6: Interactive Entry Program for Solving the 1D Schrödinger Equation

```
Input: choice (str): User selection for potential type.
       N (int): Number of basis functions.
       v (Array[float]): Fixed or variable basis widths.
       s (Array[float]): Basis center positions.
       num_levels (int) [default=3]: Number of energy levels to compute.
Output: energies (Array), optional plots
1  while True do
2      DisplayMenu ();                                    // Display menu options to the user
3      choice ← GetUserChoice ();                         // Get user's menu selection
4      if choice equals 'q' or choice equals 'Q' then
5          Print ("Program exited.");                     // Exit message
6          break;                                         // Terminate the loop
7      end
8      (N, v, s, potential_list, params_list, name_list, num_levels) ← GetParameters (choice);
       // Option 1: Custom parameters for $V(x) = x^2$
       // Option 2: Custom parameters for $V(x) = x^4 - x^2$
       // Option 3: Use default parameters for both problems
       // Option 4: Custom polynomial potential up to degree 4
9      for $i$ ← 1 to Length(potential_list) do
10         (potential, params, potential_name) ← (potential_list [$i$], params_list [$i$], name_list [$i$]);
           // Get potential function and its parameters
11         (H, S) ← BuildMatrices (N, v, s, potential, params);         // Construct H and S matrices
12         (energies, states) ← SolveSchrodinger (H, S, num_levels);    // Solve the eigenvalue problem
13         PrintEnergies (energies, potential_name, num_levels);        // Output energy levels
14         if AskUserToPlot (potential_name) then
15             x_vals ← Range ($-5$, 5, 200);
16             wavefunctions ← InitializeArray (num_levels);    // Prepare storage for wavefunctions
17             for $n$ ← 1 to num_levels do
18                 $\psi_n$ ← ComputeWavefunction (x_vals, states [:, $n$], v, s, N);   // Compute wavefunction $\psi_n$
19                 wavefunctions [$n$] ← NormalizeWavefunction (x_vals, $\psi_n$) ;   // Using trapezoidal method
20             end
21             PlotWavefunctions (x_vals, wavefunctions, num_levels, potential_name, params)
22         end
23     end
24 end
```

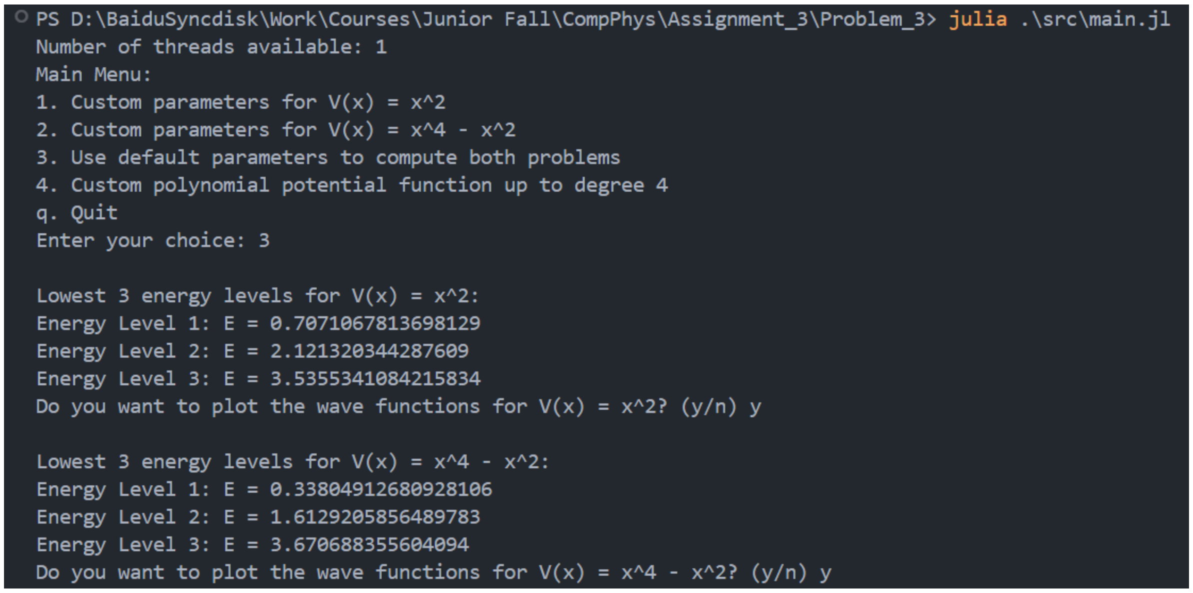
Algorithm 7: Integral Functions for Gaussian Basis

```
Function OverlapIntegral($v_1$, $s_1$, $v_2$, $s_2$):
    $v_p \leftarrow v_1 + v_2$;                                               // Combined width parameter
    $exponent, prefactor \leftarrow -\frac{v_1 v_2 (s_1 - s_2)^2}{v_p}, \frac{\sqrt{v_1 v_2}}{\sqrt{\pi v_p}}$;      // Exponent and prefactor
    $S_{ij} \leftarrow prefactor \times \mathrm{e}^{exponent}$;                 // Compute overlap integral
    return $S_{ij}$;
end
Function KineticIntegral($v_1$, $s_1$, $v_2$, $s_2$):
    $v_p \leftarrow v_1 + v_2$;                                               // Combined width parameter
    $exponent, prefactor \leftarrow -\frac{v_1 v_2 (s_1 - s_2)^2}{v_p}, \frac{v_1^{1.5} v_2^{1.5}}{\sqrt{\pi} {v_p}^{2.5}}$;   // Exponent and prefactor
    $numerator \leftarrow v_p - 2 v_1 v_2 (s_1 - s_2)^2$;                      // Numerator in kinetic integral
    $T_{ij} \leftarrow prefactor \times numerator \times \mathrm{e}^{exponent}$;  // Compute kinetic energy integral
    return $T_{ij}$;
end
Function PotentialIntegral($v_1$, $s_1$, $v_2$, $s_2$, $n$):
    $v_p \leftarrow v_1 + v_2$;                                               // Combined width parameter
    $exponent \leftarrow -\frac{v_1 v_2 (s_1 - s_2)^2}{v_p}$;
    $S_{ij} \leftarrow \frac{\sqrt{v_1 v_2}}{\sqrt{\pi v_p}} \times \mathrm{e}^{exponent}$;     // Compute overlap integral
    $\mu \leftarrow \frac{v_1 s_1 + v_2 s_2}{v_p}$;                            // Mean of combined Gaussian
    $\sigma^2 \leftarrow \frac{1}{2 v_p}$;                                     // Variance of combined Gaussian
    $M_n \leftarrow$ ComputeMoment($n$, $\mu$, $\sigma^2$);                     // Compute the $n$-th moment
    $V_{ij} \leftarrow S_{ij} \times M_n$ ;                                   // Compute potential energy integral
    return $V_{ij}$;
end
Function ComputeMoment($n$, $\mu$, $\sigma^2$):
    if $n = 0$ then
        return 1;                         // Compute $M_n$ using moments of the normal distribution
    else
        return $\sum_{k=0}^{n} \binom{n}{k} \mu^{n-k} \sigma^k \cdot (n - k - 1)!!$;     // General formula for $M_n$
    end
end
```

Algorithm 8: Matrix Construction and Schrödinger Equation Solver

```
Function BuildMatrices($N$, $v$, $s$, $PotentialFunction$, $PotentialParams$):
    Initialize $\mathbf{H}$ and $\mathbf{S}$ as $N \times N$ zero matrices;
    for $i \leftarrow 1$ to $N$ do
        for $j \leftarrow i$ to $N$ do
            $S_{ij} \leftarrow$ OverlapIntegral($v[i]$, $s[i]$, $v[j]$, $s[j]$);
            $T_{ij} \leftarrow$ KineticIntegral($v[i]$, $s[i]$, $v[j]$, $s[j]$);
            $V_{ij} \leftarrow$ PotentialFunction ($v[i]$, $s[i]$, $v[j]$, $s[j]$, PotentialParams);
            $\mathbf{H}\ [i, j] \leftarrow T_{ij} + V_{ij}$;
            $\mathbf{S}\ [i, j] \leftarrow S_{ij}$;
            if $i \neq j$ then
                $\mathbf{H}\ [j, i]$, $\mathbf{S}\ [j, i] \leftarrow \mathbf{H}\ [i, j]$, $\mathbf{S}\ [i, j]$;    // Copy symmetric elements
            end
        end
    end
    return $\mathbf{H}$, $\mathbf{S}$;
end
Function SolveSchrodinger($\mathbf{H}$, $\mathbf{S}$, $num\_levels$):
    $(E\_values, E\_vectors) \leftarrow$ GeneralizedEigenSolve($\mathbf{H}$, $\mathbf{S}$);    // Solve $\mathbf{Hc} = E\mathbf{Sc}$
    $idx \leftarrow$ Indices that sort $E\_values$ in ascending order;
    $energies \leftarrow E\_values[idx[1..num\_levels]]$;    // Select lowest $num\_levels$ energies
    $states \leftarrow E\_vectors[:, idx[1..num\_levels]]$;    // Corresponding eigenvectors
    return $energies$, $states$;
end
```

3.4 结果示例



```
PS D:\BaiduSyncdisk\Work\Courses\Junior Fall\CompPhys\Assignment_3\Problem_3> julia .\src\main.jl
Number of threads available: 1
Main Menu:
1. Custom parameters for V(x) = x^2
2. Custom parameters for V(x) = x^4 - x^2
3. Use default parameters to compute both problems
4. Custom polynomial potential function up to degree 4
q. Quit
Enter your choice: 3

Lowest 3 energy levels for V(x) = x^2:
Energy Level 1: E = 0.7071067813698129
Energy Level 2: E = 2.121320344287609
Energy Level 3: E = 3.5355341084215834
Do you want to plot the wave functions for V(x) = x^2? (y/n) y

Lowest 3 energy levels for V(x) = x^4 - x^2:
Energy Level 1: E = 0.33804912680928106
Energy Level 2: E = 1.6129205856489783
Energy Level 3: E = 3.670688355604094
Do you want to plot the wave functions for V(x) = x^4 - x^2? (y/n) y
```

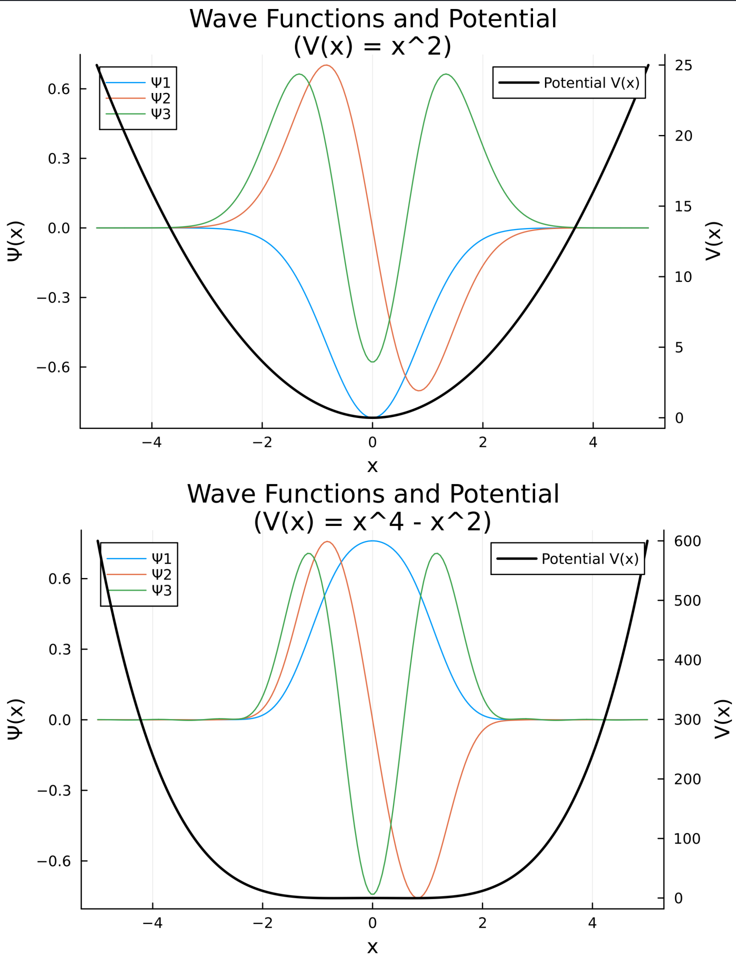
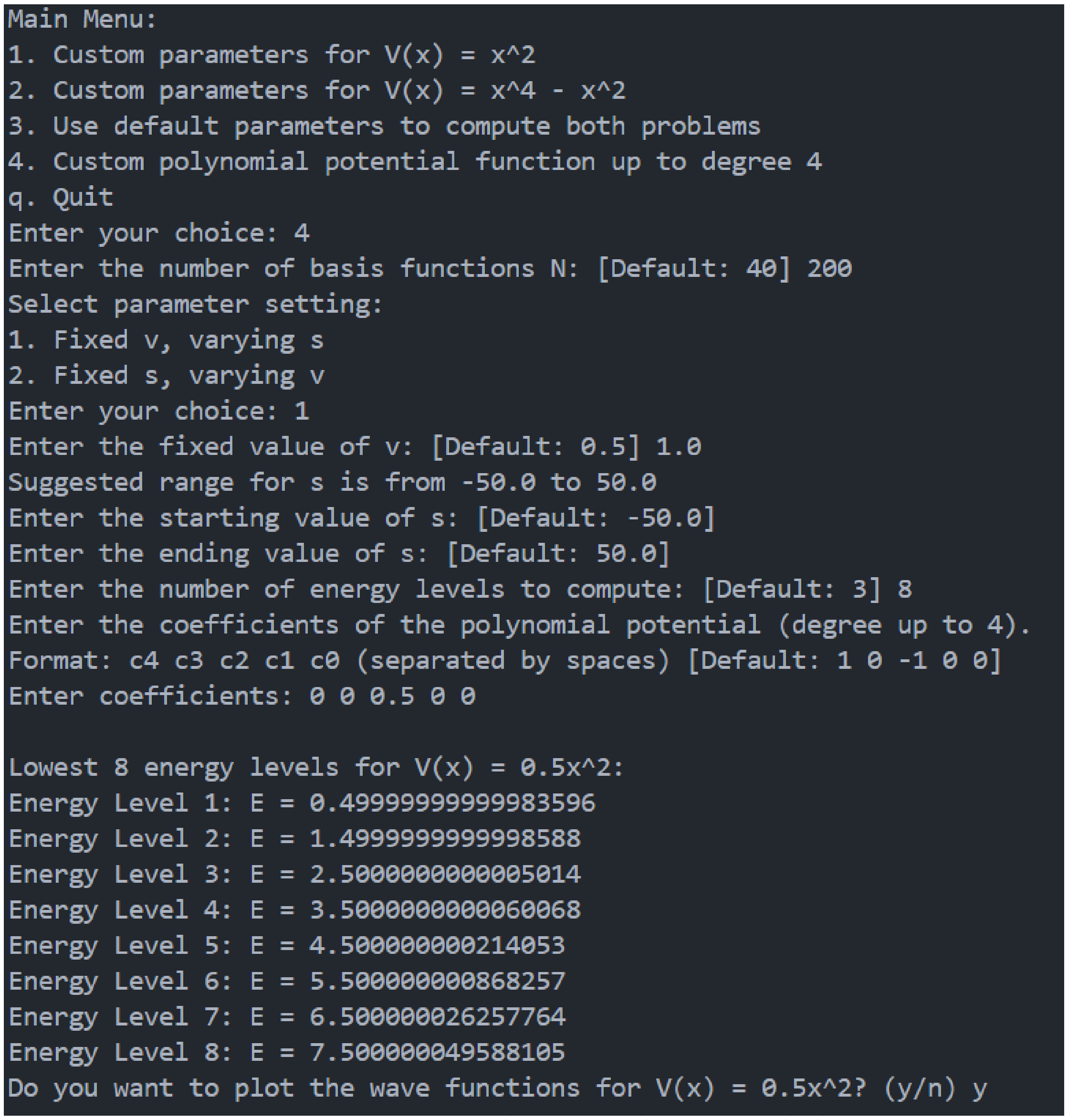


图 6: 选项 3：使用默认参数求解 $V = x^2$ 与 $V = x^4 - x^2$ 及其前 3 个能级



```
Main Menu:
1. Custom parameters for V(x) = x^2
2. Custom parameters for V(x) = x^4 - x^2
3. Use default parameters to compute both problems
4. Custom polynomial potential function up to degree 4
q. Quit
Enter your choice: 4
Enter the number of basis functions N: [Default: 40] 200
Select parameter setting:
1. Fixed v, varying s
2. Fixed s, varying v
Enter your choice: 1
Enter the fixed value of v: [Default: 0.5] 1.0
Suggested range for s is from -50.0 to 50.0
Enter the starting value of s: [Default: -50.0]
Enter the ending value of s: [Default: 50.0]
Enter the number of energy levels to compute: [Default: 3] 8
Enter the coefficients of the polynomial potential (degree up to 4).
Format: c4 c3 c2 c1 c0 (separated by spaces) [Default: 1 0 -1 0 0]
Enter coefficients: 0 0 0.5 0 0

Lowest 8 energy levels for V(x) = 0.5x^2:
Energy Level 1: E = 0.49999999999983596
Energy Level 2: E = 1.4999999999998588
Energy Level 3: E = 2.5000000000005014
Energy Level 4: E = 3.5000000000060068
Energy Level 5: E = 4.500000000214053
Energy Level 6: E = 5.500000000868257
Energy Level 7: E = 6.500000026257764
Energy Level 8: E = 7.500000049588105
Do you want to plot the wave functions for V(x) = 0.5x^2? (y/n) y
```

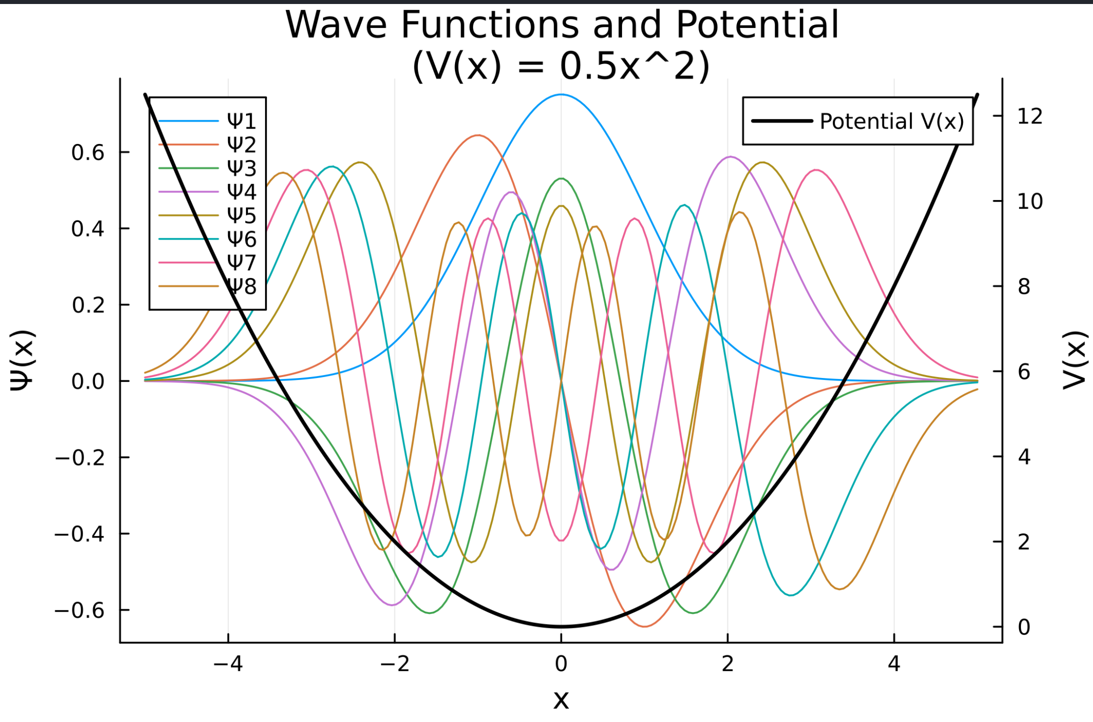
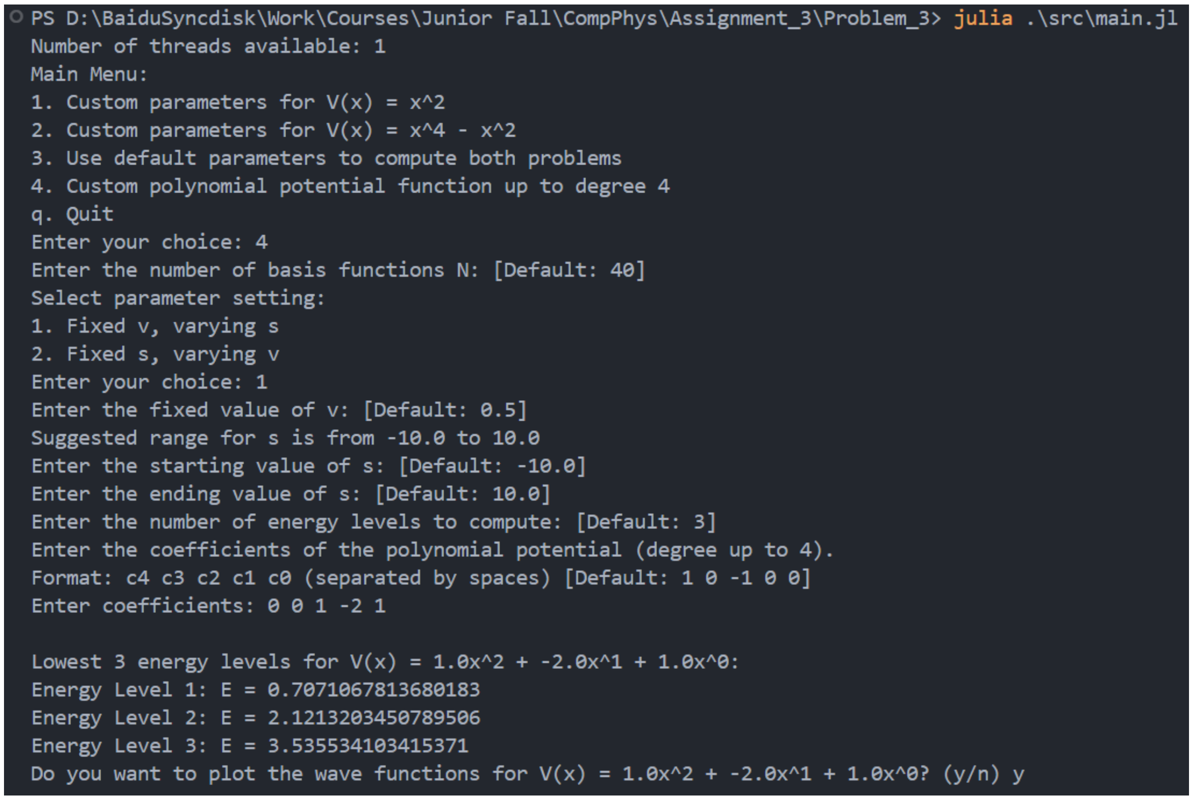


图 7: 选项 4：求解自定义势阱 $V = \frac{1}{2}x^2$ 及其前 8 个能级

注： 在自然单位制下，与解析解 $E_n = (n + \frac{1}{2})\hbar\omega$ 数值一致



```
PS D:\BaiduSyncdisk\Work\Courses\Junior Fall\CompPhys\Assignment_3\Problem_3> julia .\src\main.jl
Number of threads available: 1
Main Menu:
1. Custom parameters for V(x) = x^2
2. Custom parameters for V(x) = x^4 - x^2
3. Use default parameters to compute both problems
4. Custom polynomial potential function up to degree 4
q. Quit
Enter your choice: 4
Enter the number of basis functions N: [Default: 40]
Select parameter setting:
1. Fixed v, varying s
2. Fixed s, varying v
Enter your choice: 1
Enter the fixed value of v: [Default: 0.5]
Suggested range for s is from -10.0 to 10.0
Enter the starting value of s: [Default: -10.0]
Enter the ending value of s: [Default: 10.0]
Enter the number of energy levels to compute: [Default: 3]
Enter the coefficients of the polynomial potential (degree up to 4).
Format: c4 c3 c2 c1 c0 (separated by spaces) [Default: 1 0 -1 0 0]
Enter coefficients: 0 0 1 -2 1

Lowest 3 energy levels for V(x) = 1.0x^2 + -2.0x^1 + 1.0x^0:
Energy Level 1: E = 0.7071067813680183
Energy Level 2: E = 2.1213203450789506
Energy Level 3: E = 3.535534103415371
Do you want to plot the wave functions for V(x) = 1.0x^2 + -2.0x^1 + 1.0x^0? (y/n) y
```

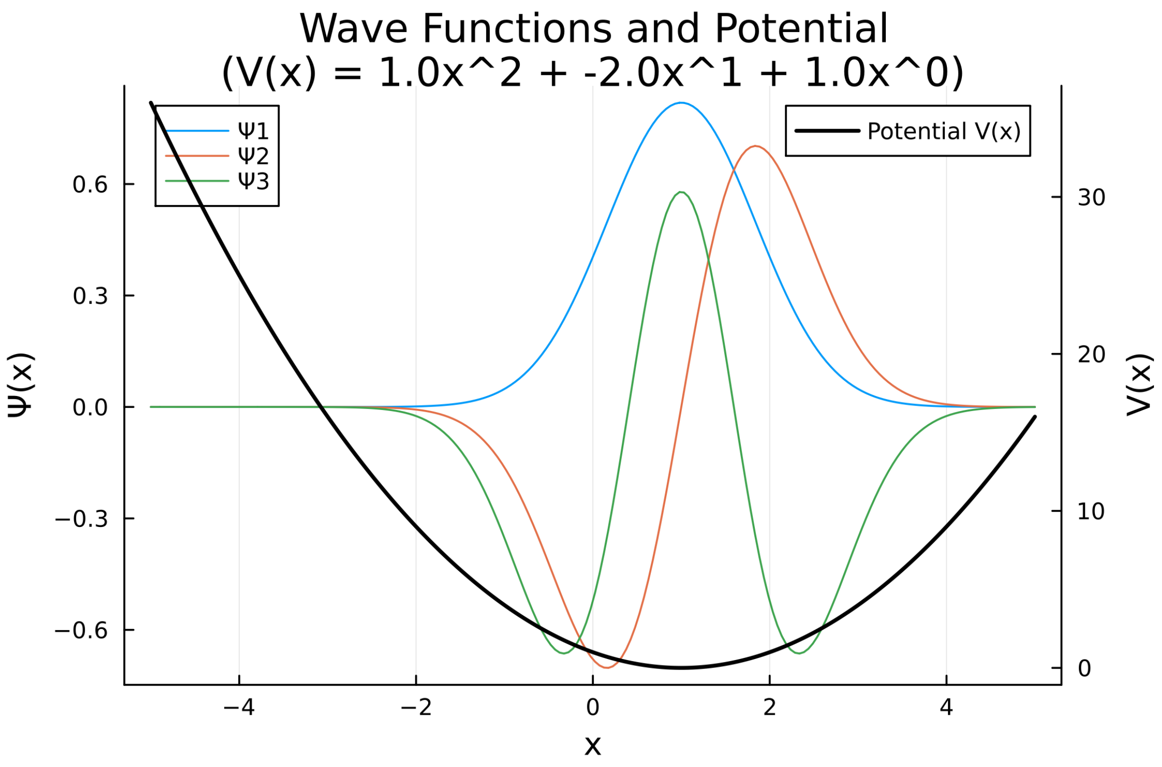


图 8: 选项 4：求解自定义势阱 $V = (x+1)^2$ 及其前 3 个能级

注： 能级与势阱 $V = x^2$ 一致，波函数相较势阱 $V = x^2$ 平移