线性方程组 AX=B 的数值计算方法实现

姓名：王旋宇
学号：2012049010022
班级：数理基科班
老师：赖生建

实验目的

一、掌握高斯消元法和选主元策略，会编程实现算法。
二、掌握三角分解法，将矩阵进行 LU 分解并求解。会编程实现矩阵的 LU 分解及求解。
三、了解雅可比迭代。掌握高斯-赛德尔迭代，并会编程实现且求解。

实验原理

一、高斯消去法和选主元

1. 定理 1（初等变换） 下面三种变换可是一个线性方程组变换成另一个等价的线性方程组：
   - Interchange（交换）变换：对调方程组的两行。
   - Scaling（比例）变换：用非零数乘以方程组的某一行。
   - Replacement（置换）变换：将方程组的某一行乘以一个非零常数，再加到另一行上。
2. 定理 2（初等行变换） 对增广矩阵进行如下变换可得到一个等价的线性方程组。
   - Interchange（交换）变换：对调矩阵的两行。
   - Scaling（比例）变换：用非零数乘以矩阵的某一行的所有元素。
   - Replacement（置换）变换：将矩阵的某一行的所有元素乘以一个非零常数，再加到另一行对应的元素上，即：

$$row_r = row_r - m_{rp} \times row_p \tag{1}$$

3. 定义 1 系数矩阵 A 中的元素a_{rr}用来消去a_{kr}，其中$k = r+1, r+2, \cdots, N$，这里称a_{rr}为第 r 个主元，第 r 行被称为主元行。
4. 高斯消元法。高斯消元法就是将矩阵反复进行行初等变换，将矩阵变为行阶梯型矩阵，即将矩阵变为上三角矩阵。之后利用回代法求解。
5. 偏序选主元策略。在高斯消元法消元的消元过程中，如果出现约化主元 $a_{pp}^{(p)}$为 0 的情况，消元过程将无法进行。有时即使约化主元不为零，但其绝对值很小也会导致计算数据的增长或舍入误差扩散。这时候有必要寻找更合适的第 k 行的$a_{kp}^{(p)}$，然后交换第 k 行和第 p 行。选择行的判定条件称为选主元策略。偏序选主元策略是最常用的选主元策略。

   偏序选主元策略首先检查位于主对角线或主对角线下方第 p 列的所有元素，确定行 k，它的元素的绝对值最大。如果 K>p，则交换第 k 行和第 p 行。这样保证了消元后的矩阵和初始系数矩阵的对应元素的相对大小一致。

二、三角分解法

1. 定义 2　如果非奇异矩阵 A 可表示为下三角矩阵 L 和上三角矩阵 U 的乘积：
   $A = LU$(2).
   则 A 存在一个三角分解。
2. 设线性方程组AX = B的系数矩阵 A 存在三角分解，则线性方程组可表示为
   $LUX = B$(3).
   而方程组的解可通过定义$Y = UX$并求解$LY = B$和$UX = Y$得到 X。
3. LU 矩阵的求法。设需要将矩阵 A 进行 LU 分解，A 满足在求解过程中无行交换过程。那么在第 i 行进行消元的时候，同时将消元倍数放在一个单位矩阵的对应位置。在 A 消元结束之后，单位矩阵就变成了 L，同时 A 就变成了 U。
   拓展的 LU 矩阵求法。在上面的 LU 求法中要求 A 无需进行行变换，这样有可能使得一个非奇异矩阵 A 不能直接分解为$A = LU$，于是有如下定理。
   定理 3　如果 A 是非奇异矩阵，则存在一个置换矩阵 P，使得 PA 存在三角分解。

三、求解线性方程组的迭代法。

1. 雅可比迭代。
   $$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k)} - \cdots - a_{jj-1}x_{j-1}^{(k)} - a_{jj+1}x_{j+1}^{(k)} - \cdots - a_{jN}x_N^{(k)}}{a_{jj}}, j = 1,2,\cdots,N \quad (4).$$
   其中，矩阵为$N \times N$维矩阵。
2. 高斯-赛德尔迭代。
   $$x_j^{(k+1)} = \frac{b_j - a_{j1}x_1^{(k+1)} - \cdots - a_{jj-1}x_{j-1}^{(k+1)} - a_{jj+1}x_{j+1}^{(k)} - \cdots - a_{jN}x_N^{(k)}}{a_{jj}}, j = 1,2,\cdots,N \quad (5).$$
   其中，矩阵为$N \times N$维矩阵。
3. 定义 3　设有$N \times N$维矩阵 A，如果
   $$|a_{kk}| > \sum_{\substack{j=1 \\ j \neq k}}^{N} |a_{kj}|, k = 1,2,\cdots,N \quad (6).$$
   则称 A 具有严格对角优势。
4. 定理 4（雅可比迭代和高斯-赛德尔迭代）　设矩阵A具有严格对角优势，则AX = B有为一解$X = P$。利用迭代式可产生一个向量序列$\{P_k\}$，而且对于任意初始向量P_0，向量序列都将收敛到P。
5. 定义 4　定义范数$\|*\|_1$为：
   $$\|X\|_1 = \sum_{j=1}^{N} |x_j|。 \quad (7).$$
   这个范数又称为列和范数。可以用来比较两个向量之间的差距，相比欧几里得距离具有计算量小的优点。可以用来判断向量序列的收敛。

实验内容

1. P108 1. 许多科学应用包含的矩阵带有很多零，在实际情况中很重要的三角型线性方程组有如下形式：

$$
\begin{array}{lllllllll}
d_1x_1 & c_1x_2 & & & & & & = & b_1 \\
a_1x_1 & d_2x_2 & c_2x_3 & & & & & = & b_2 \\
 & a_2x_2 & d_3x_3 & c_3x_4 & & & & = & b_3 \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & & & a_{N-2}x_{N-2} & d_{N-1}x_{N-1} & c_{N-1}x_N & = & b_{N-1} \\
 & & & & & a_{N-1}x_{N-1} & d_Nx_N & = & b_N
\end{array} \quad (8).
$$

构造一个程序求解三角型线性方程组。可假定不需要行变换，而且可用第 k 行消去第 k+1 行的x_k。

2. P120 1. 使用程序 3.3 求解线性方程组$AX = B$。其中

$$
A = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 2 & -1 & 3 & 5 \\ 0 & 0 & 2 & 5 \\ -2 & -6 & -3 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad (9).
$$

使用 MATLAB 中的[L,U,P]=lu(A)命令检查得到的答案。

3. P120 2. 使用程序 3.3 求解线性方程组$AX = B$，其中$A = [a_{ij}]_{N\times N}, a_{ij} = i^{j-1}$；而且$B = [b_{ij}]_{N\times 1}, b_{11} = 1$, 当 i ≥ 2 时，$b_{1j} = (i^j - 1)/(i - 1)$。对$N = 3,7,11$的情况分别求解。精确解为$X = [1\ 1\ \cdots 1\ 1]'$。对得到的结果与精确解的差异进行解释。

4. P120 3. 修改程序 3.3，使得它可以通过重复求解 N 个线性方程组

$$
AC_j = E_j, \text{其中} j = 1,2,\cdots,N \quad (10).
$$

来得到A^{-1},
则

$$
A[C_1\ C_2\ \cdots\ C_N] = [E_1\ E_2\ \cdots\ E_N] \quad (11).
$$

而且

$$
\mathrm{A}^{-1} = [C_1\ C_2\ \cdots\ C_N] \quad (12).
$$

保证对 LU 分解只计算一次！

5. P129 3. 设有如下三角线性方程组，而且系数矩阵具有严格对角优势：

$$
\begin{array}{lllllllll}
d_1x_1 & c_1x_2 & & & & & & = & b_1 \\
a_1x_1 & d_2x_2 & c_2x_3 & & & & & = & b_2 \\
 & a_2x_2 & d_3x_3 & c_3x_4 & & & & = & b_3 \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & \cdots & \cdots & \cdots & & & = & \cdots \\
 & & & & a_{N-2}x_{N-2} & d_{N-1}x_{N-1} & c_{N-1}x_N & = & b_{N-1} \\
 & & & & & a_{N-1}x_{N-1} & d_Nx_N & = & b_N
\end{array} \quad (13).
$$

i). 根据方程组（9）、式（10）和式（11），设计一个算法来求解上述方程组。算法必须有效的利用系数矩阵的稀疏性。

ii). 根据 i)中设计的算法构造一个 MATLAB 程序，并求解下列三角线性方程组。

a)
$$\begin{array}{ccccccccc} 4m_1 & + & m_2 & & & = & 3 \\ m_1 & + & 4m_2 & + & m_3 & = & 3 \\ m_2 & + & 4m_3 & + & m_4 & = & 3 \\ m_3 & + & 4m_4 & + & m_5 & = & 3 \\ \vdots & & \vdots & & \vdots & & \vdots \\ m_{48} & + & m_{49} & + & m_{50} & = & 3 \\ & & m_{49} & + & m_{50} & = & 3 \end{array} \quad (14).$$

b)
$$\begin{array}{ccccccccc} 4m_1 & + & m_2 & & & = & 1 \\ m_1 & + & 4m_2 & + & m_3 & = & 2 \\ m_2 & + & 4m_3 & + & m_4 & = & 1 \\ m_3 & + & 4m_4 & + & m_5 & = & 2 \\ \vdots & & \vdots & & \vdots & & \vdots \\ m_{48} & + & m_{49} & + & m_{50} & = & 1 \\ & & m_{49} & + & m_{50} & = & 2 \end{array} \quad (15).$$

6. P130 4. 利用高斯-赛德尔迭代法求解下列带状方程。

$$\begin{array}{ccccccccccc} X_1 & + & X_1 & + & X_1 & & & & & = & 5 \\ X_1 & + & X_1 & + & X_1 & + & X_1 & & & = & 5 \\ X_1 & + & X_1 & + & X_1 & + & X_1 & + & X_1 & = & 5 \\ X_1 & + & X_1 & + & X_1 & + & X_1 & + & X_1 & = & 5 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ X_1 & + & X_1 & + & X_1 & + & X_1 & + & X_1 & = & 5 \\ & & X_1 & + & X_1 & + & X_1 & + & X_1 & = & 5 \\ & & & & X_1 & + & X_1 & + & X_1 & = & 5 \end{array} \quad (16).$$

实验分析

1. P108 1. 观察矩阵可以知道只要从第一行开始，用每一行来消去下一行的首元，这样得到的就是矩阵中只剩下a_{kk}与a_{kk+1}两个元素（除了最后一行只有x_N项）。这时可以采用从最后一行向上消元的办法来解方程组。不过也可以采用回代法解方程组。本道题目中采取回代法来解方程组。程序使用 C 语言编写。
2. P120 1. 本道题目采用带选主元的三角分解法，不过采用 C 语言实现不使用程序 3.3。首先使用偏序选主元策略得到一个置换矩阵 P，然后对置换后的矩阵 A 进行 LU 分解，通过定义$Y = UX$并求解$LY = B$和$UX = Y$得到 X。

3. P120 2. 这道题的求解思路和上题一样，只是在初始化矩阵的时候改变一下初始化函数即可。
4. P120 3. 这道题采用第二题的程序。先构造一个 N 阶单位矩阵。然后对于单位矩阵的每一列都复制给一个$N \times 1$的列向量，即第二题中的 B，然后按照第二题所示方法求得解之后把解复制到单位矩阵的对应列即可。
5. P129 3. 采用高斯-赛德尔迭代方法计算。不过在计算式（）中的分子中的被减项的和的时候，利用三角线性方程组的系数特征，只计算$a_{ii-1} \cdot x_{i-1}, a_{ii} \cdot x_i, a_{ii+1} \cdot x_{i+1}$三项的和，如果某个系数不存在就略去不计算。这样在每一迭代中需要计算$(N-1) \times N$次乘积，现在变为了只用计算$3 \times (N-2) + 4$次乘积。其余的按照普通的高斯-赛德尔迭代算法计算即可。
6. P130 4. 本题目采用高斯-赛德尔迭代方法计算。仅仅按照普通的高斯-赛德尔迭代算法计算即可。并没有利用带状矩阵的系数特征。

实验结果

1. P108 1. 以解如下两个线性方程为例。

i).

$$\begin{aligned} x_1 + 2x_2 &= 7 \\ 2x_1 + 3x_2 - x_3 &= 9 \\ 4x_2 + 2x_3 + 3x_4 &= 10 \\ 2x_3 - 4x_4 &= 12 \end{aligned}$$

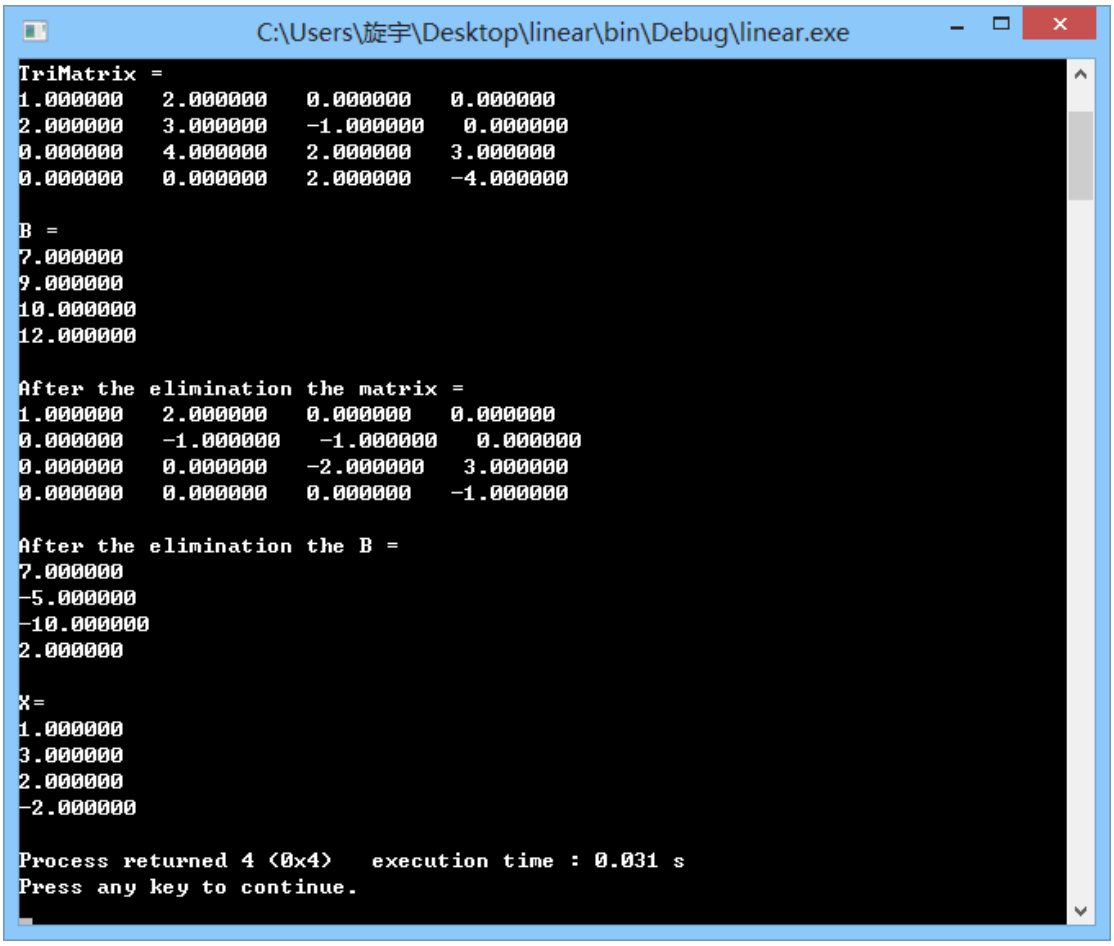

图 1.　　P108 1.题目测试数据第一组计算结果

ii).

$$\begin{aligned} x_1 + x_2 &= 5 \\ 2x_1 - x_2 + 5x_3 &= -9 \\ 3x_2 - 4x_3 + 2x_4 &= 19 \\ 2x_3 + 6x_4 &= 2 \end{aligned}$$

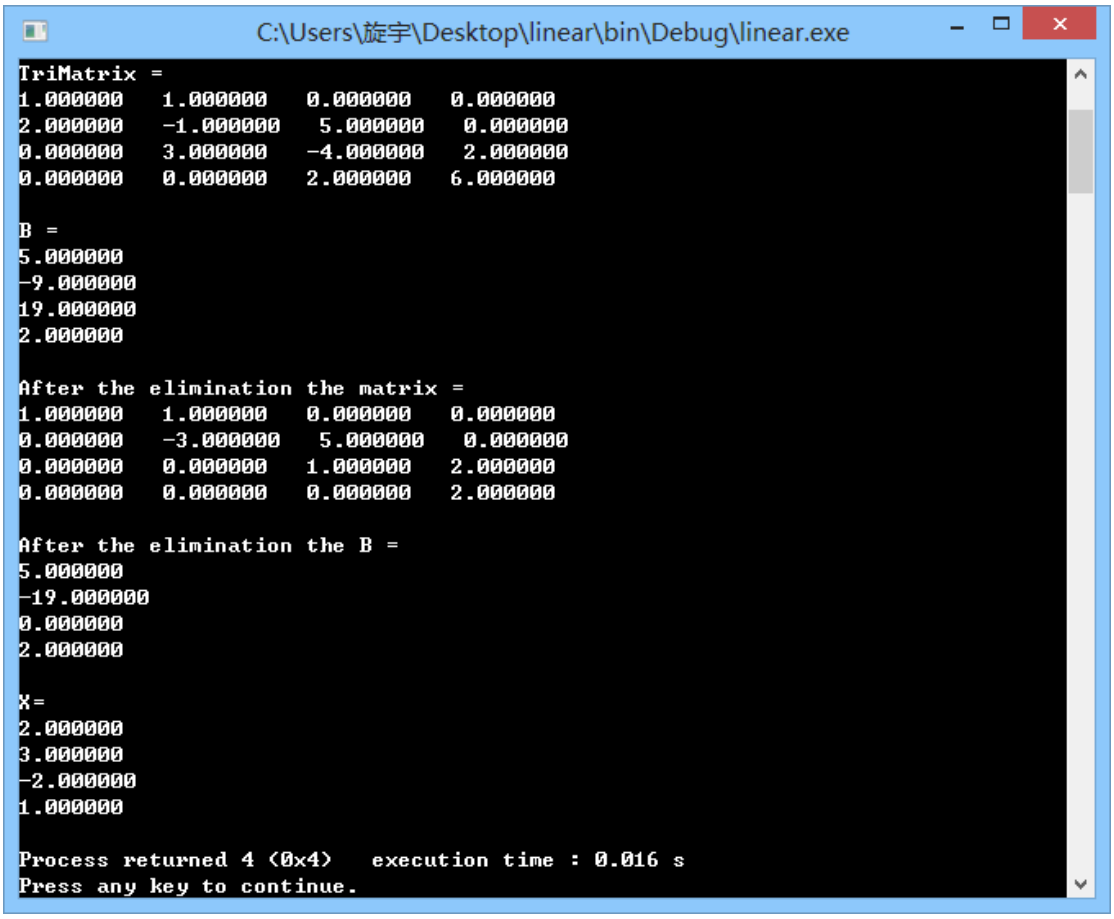

图 2.　　P108 1.题目测试数据第一组计算结果

2.　P120 1.

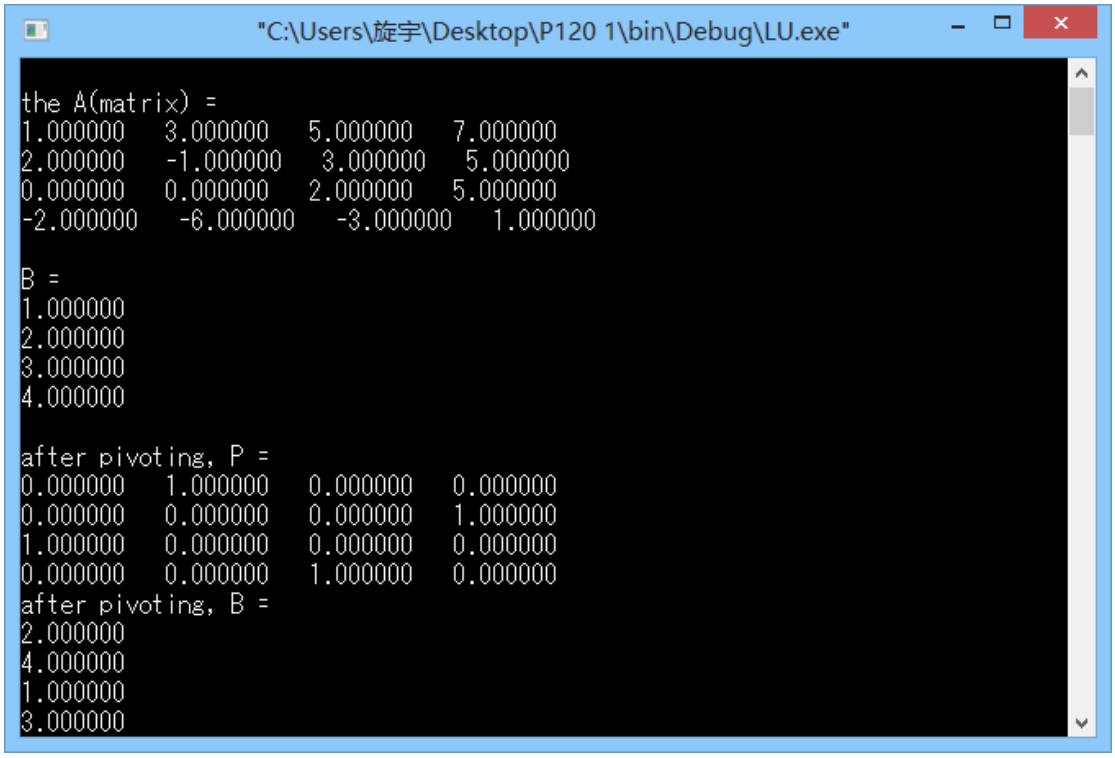

图 3. P120 1. 题目计算结果 1

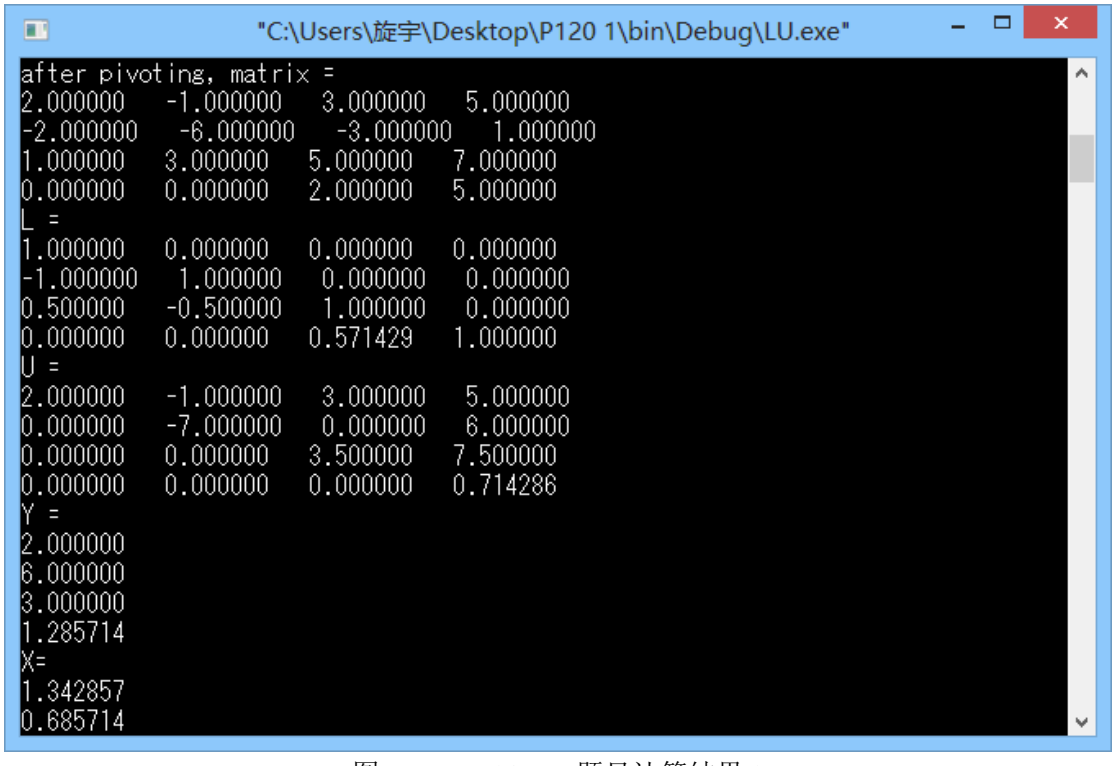

图 4. P120 1. 题目计算结果 2

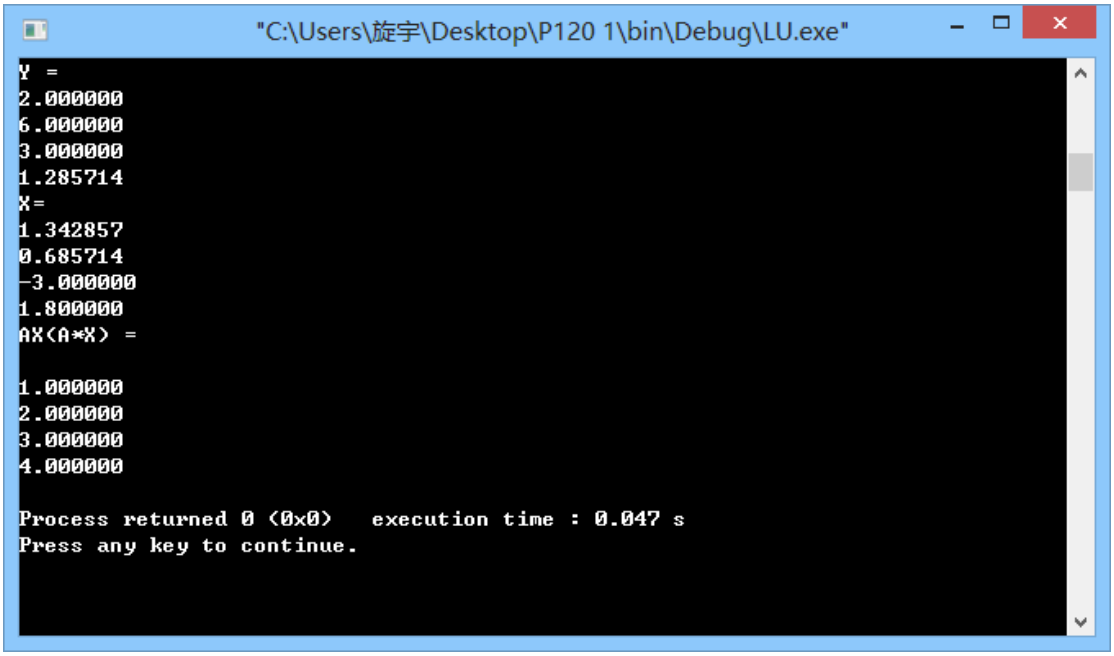

图 5. P120 1. 题目计算结果 3

```
A =

     1     3     5     7
     2    -1     3     5
     0     0     2     5
    -2    -6    -3     1

>> B

B =

     1
     2
     3
     4

>> [L,U,P]=lu(A)

L =

    1.0000         0         0         0
   -1.0000    1.0000         0         0
    0.5000   -0.5000    1.0000         0
         0         0    0.5714    1.0000
```

图 6. P120 1. 题目计算结果 MATLAB 验证 1

```
U =

    2.0000   -1.0000    3.0000    5.0000
         0   -7.0000         0    6.0000
         0         0    3.5000    7.5000
         0         0         0    0.7143


P =

     0     1     0     0
     0     0     0     1
     1     0     0     0
     0     0     1     0

>> A^(-1)*B

ans =

    1.3429
    0.6857
   -3.0000
    1.8000
```

图 7.　　P120 1.　题目计算结果 MATLAB 验证 2

3. P120 2.
   - 当 N=3 时程序运行结果

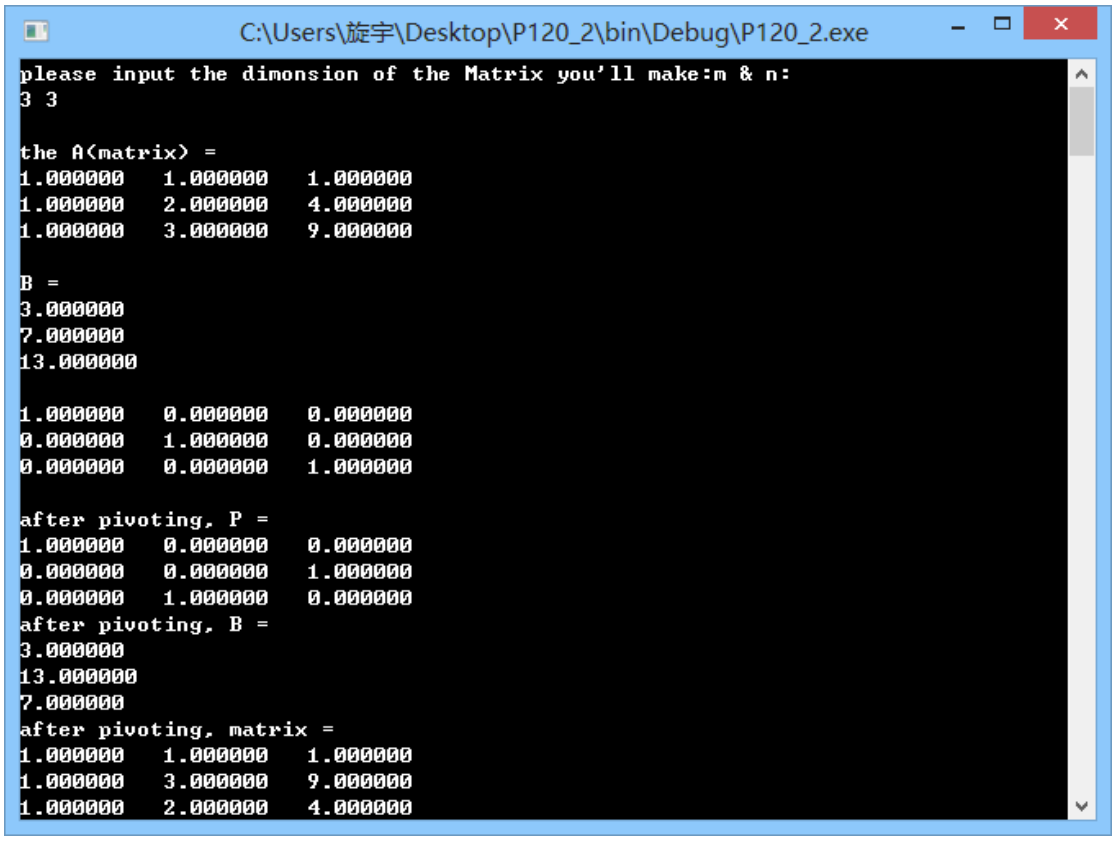

图 8.　　P120 2.　N=3 计算结果图 1

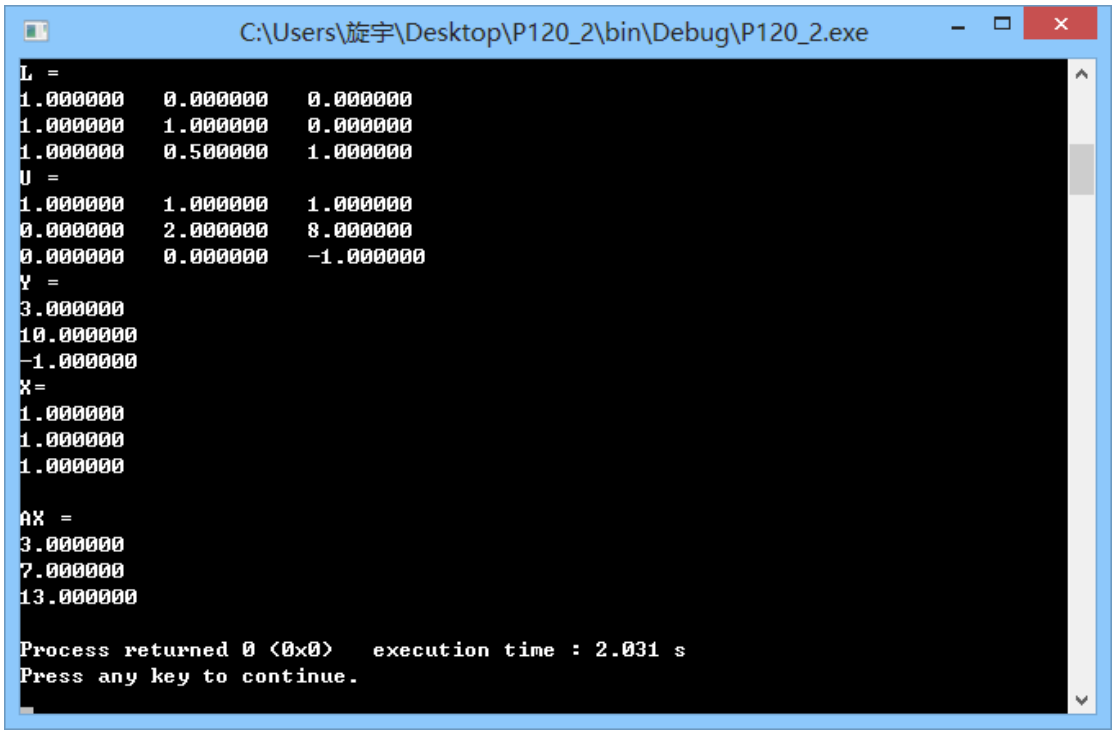

图 9.　　P120 2.　N=3 计算结果图 2

- 当 N=7 时程序运行结果

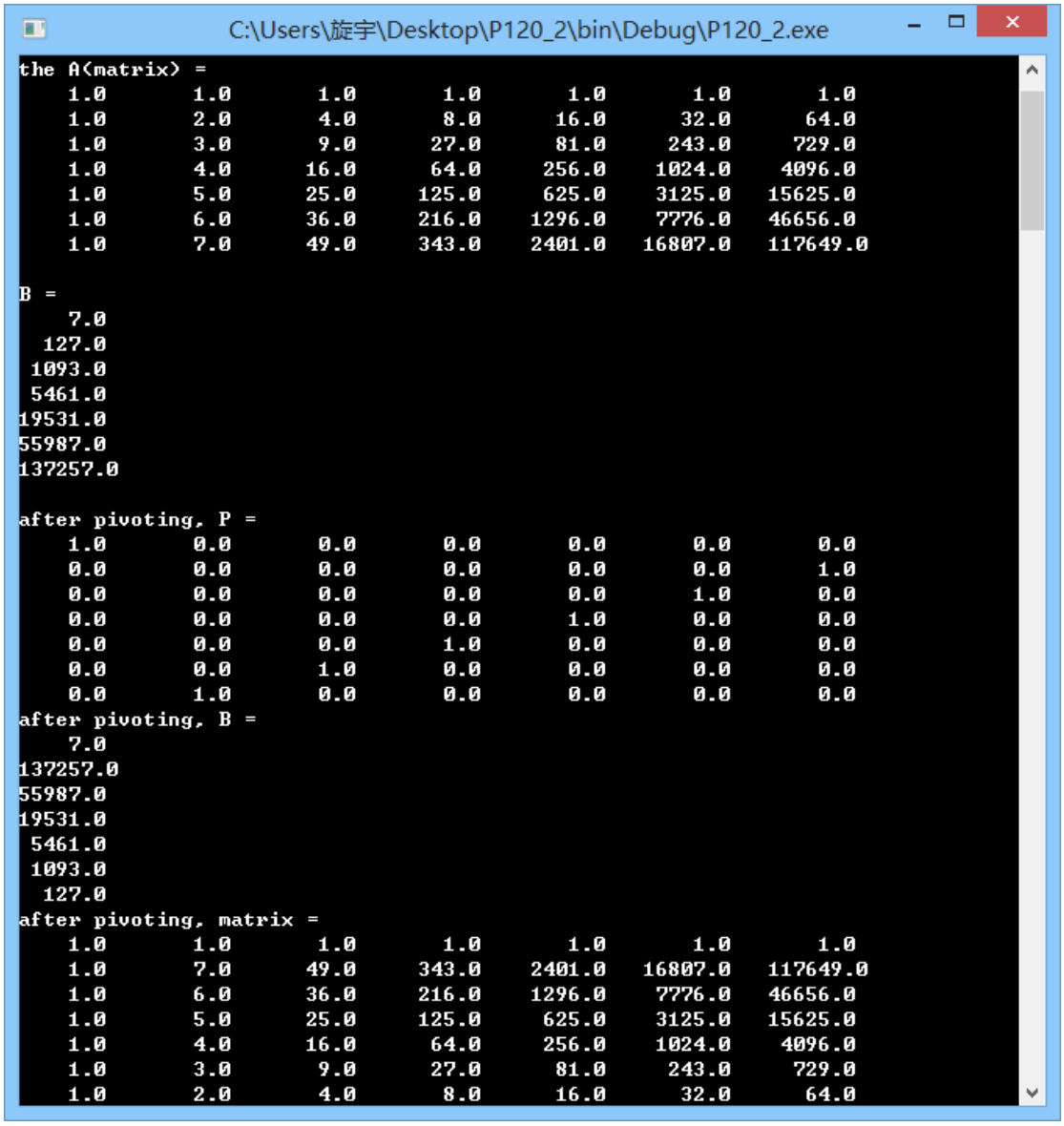

图 10.　P120 2.　N=7 计算结果图 1

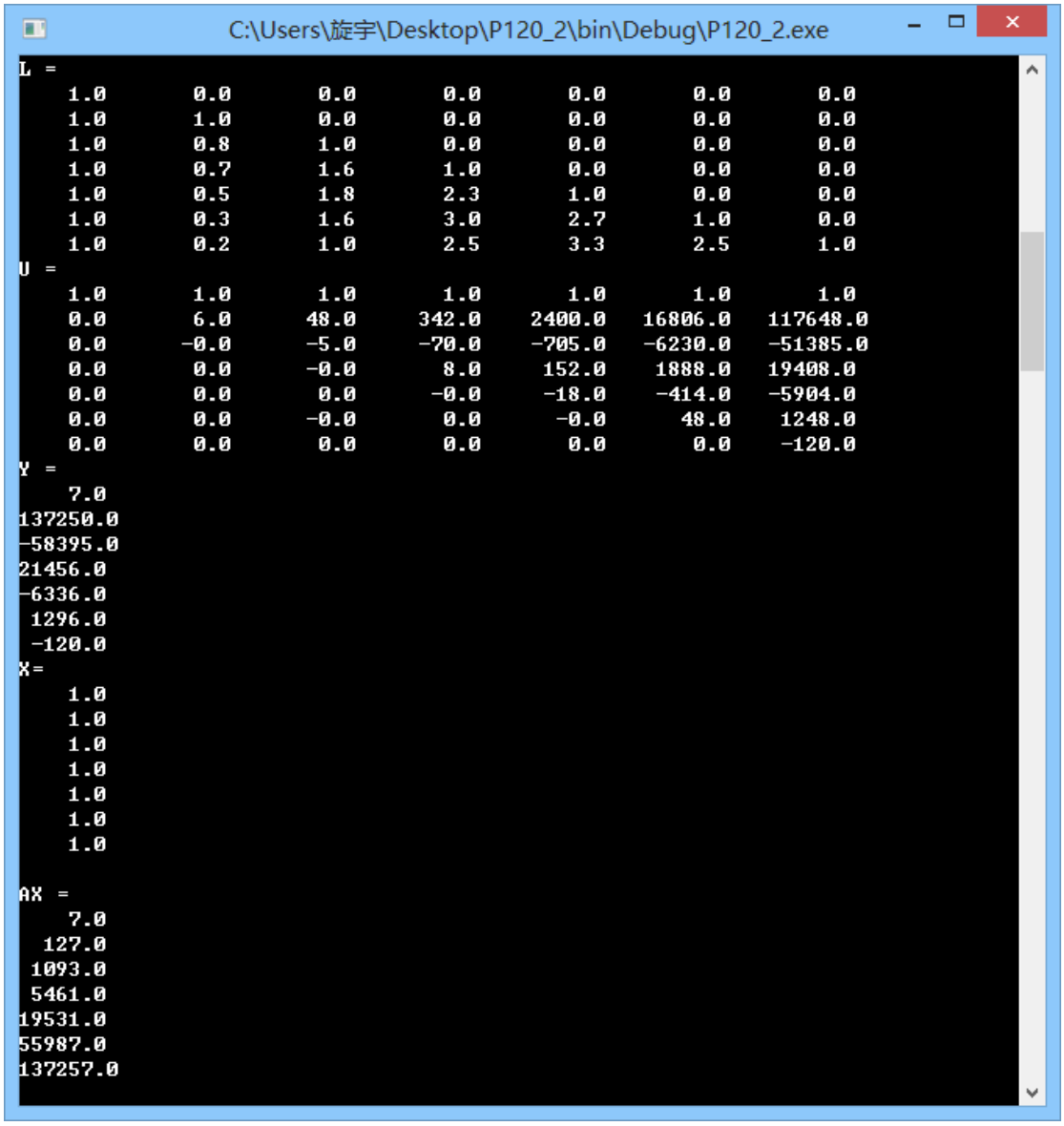

图 11. P120 2. N=7 计算结果图 2

- 当 N=11 时候的程序运行结果

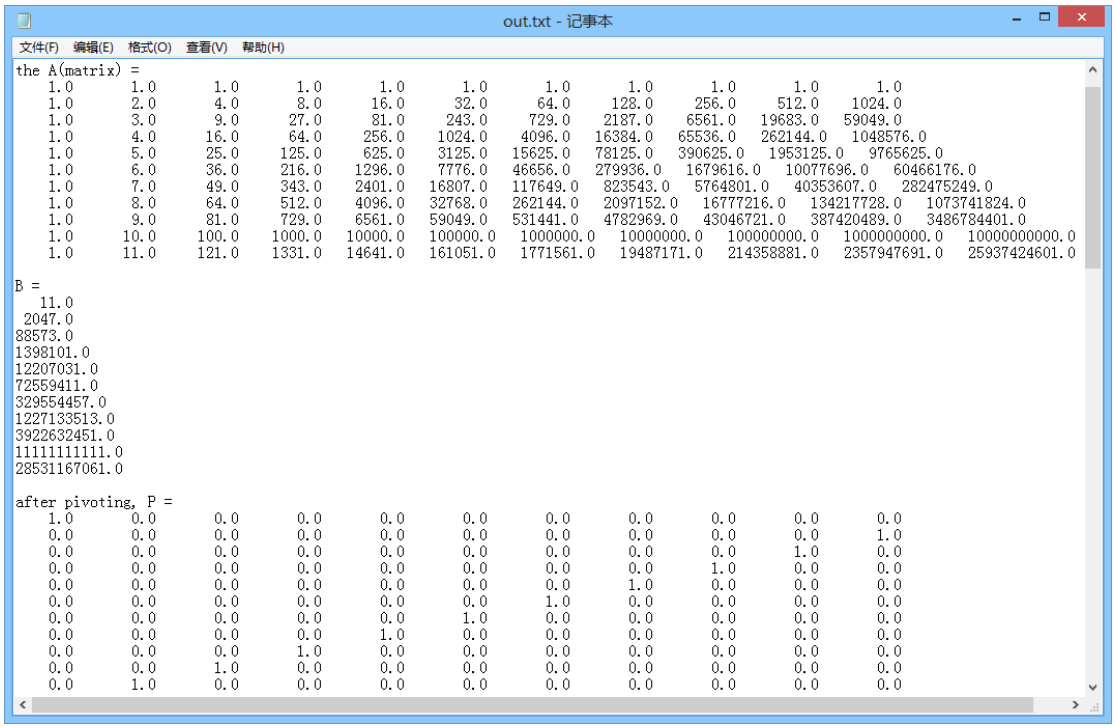

图 12. P120 2. N=11 计算结果图 1

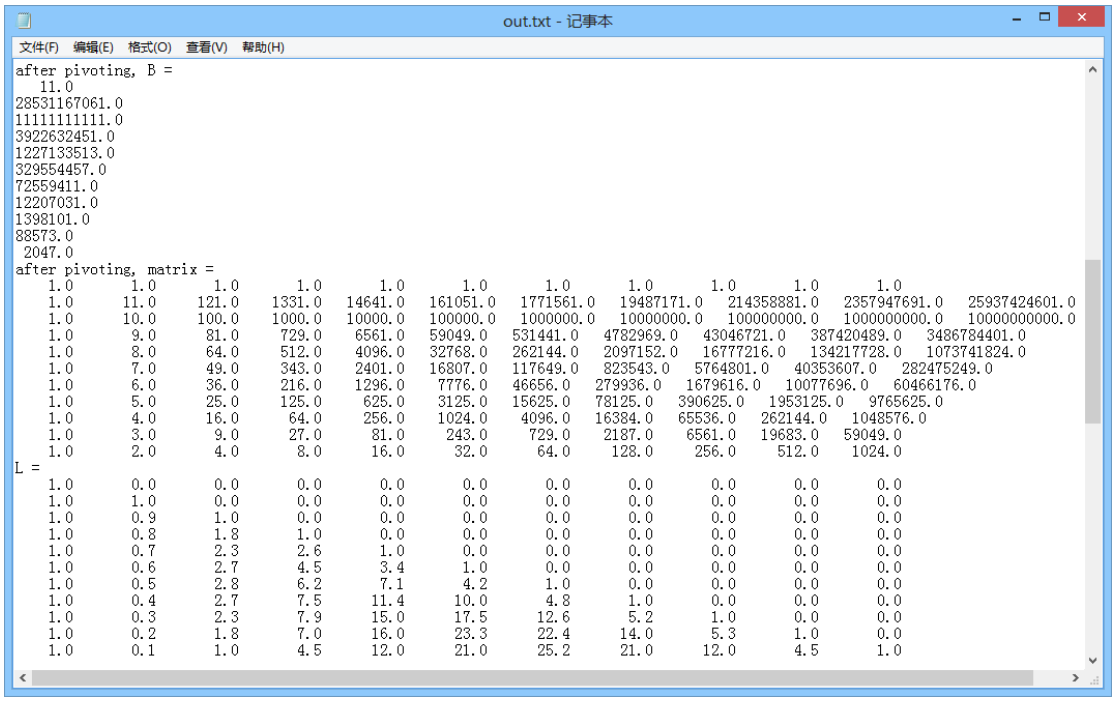

图 13. P120 2. N=11 计算结果图 2

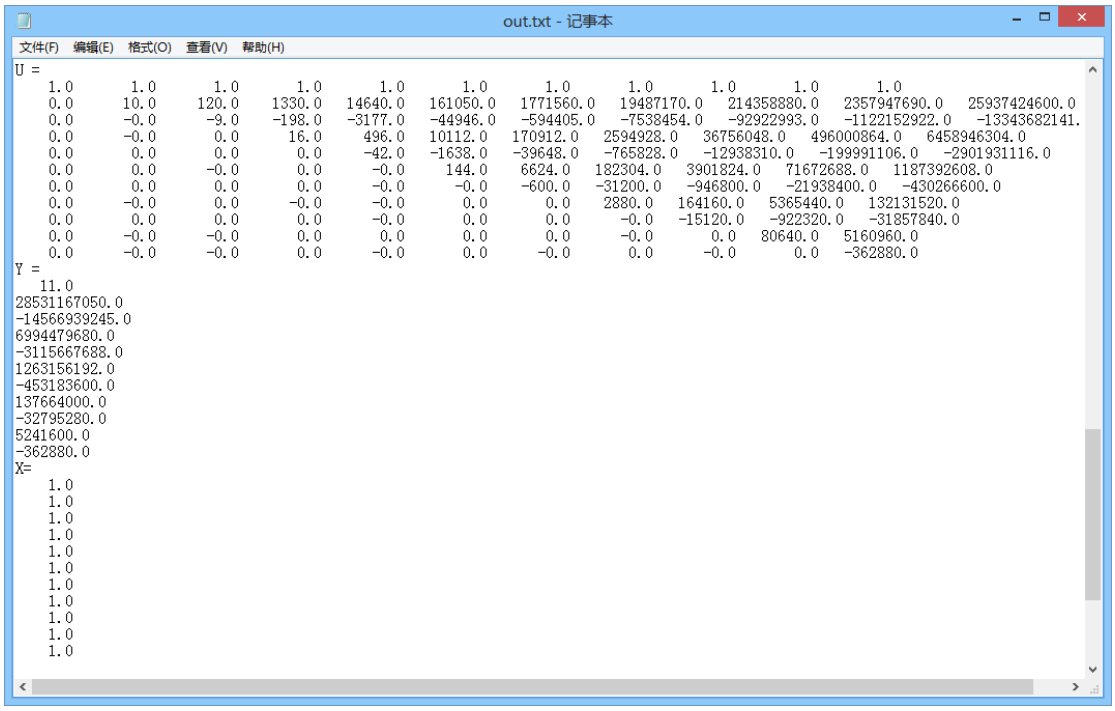

图 14. P120 2. N=11 计算结果图 3

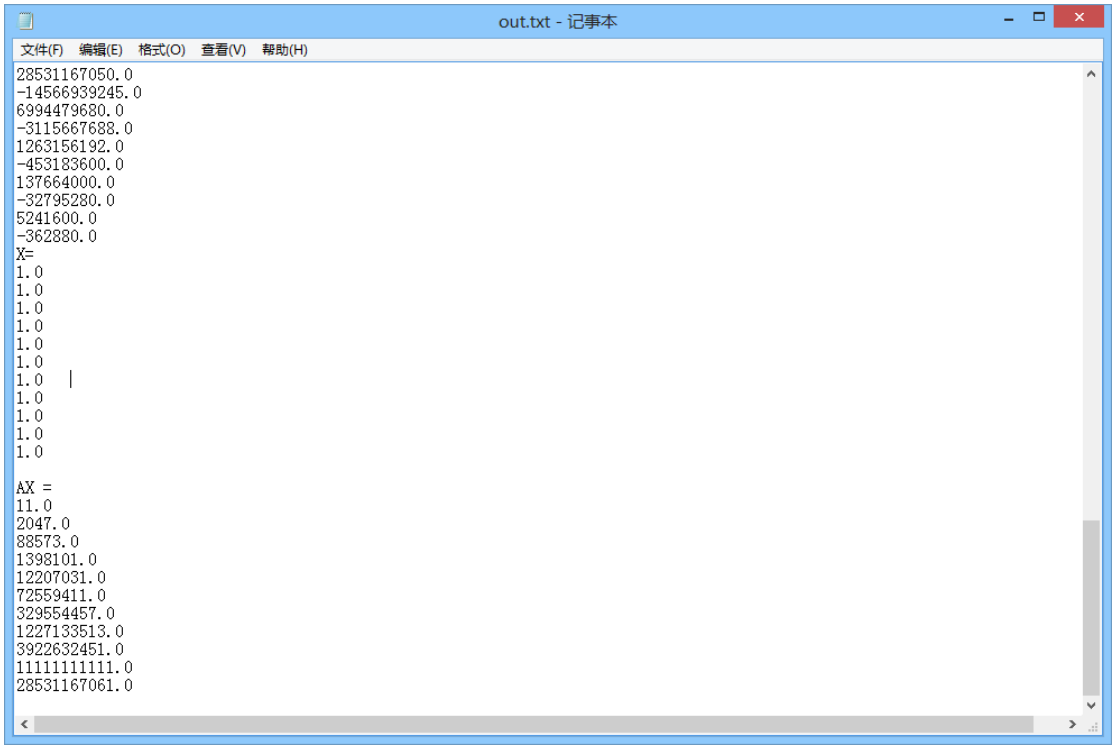

图 15. P120 2. N=11 计算结果图 4

4. P120 3.

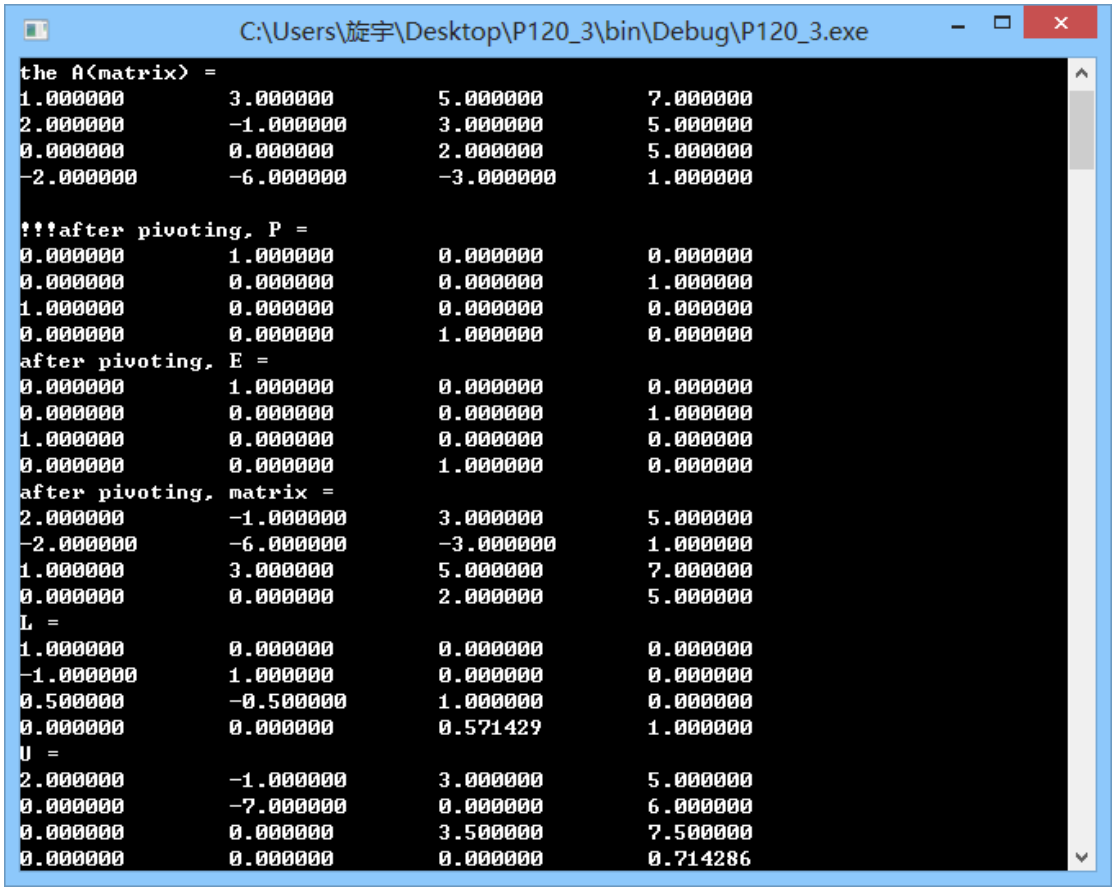

图 15.　P120 3.　矩阵分解计算结果

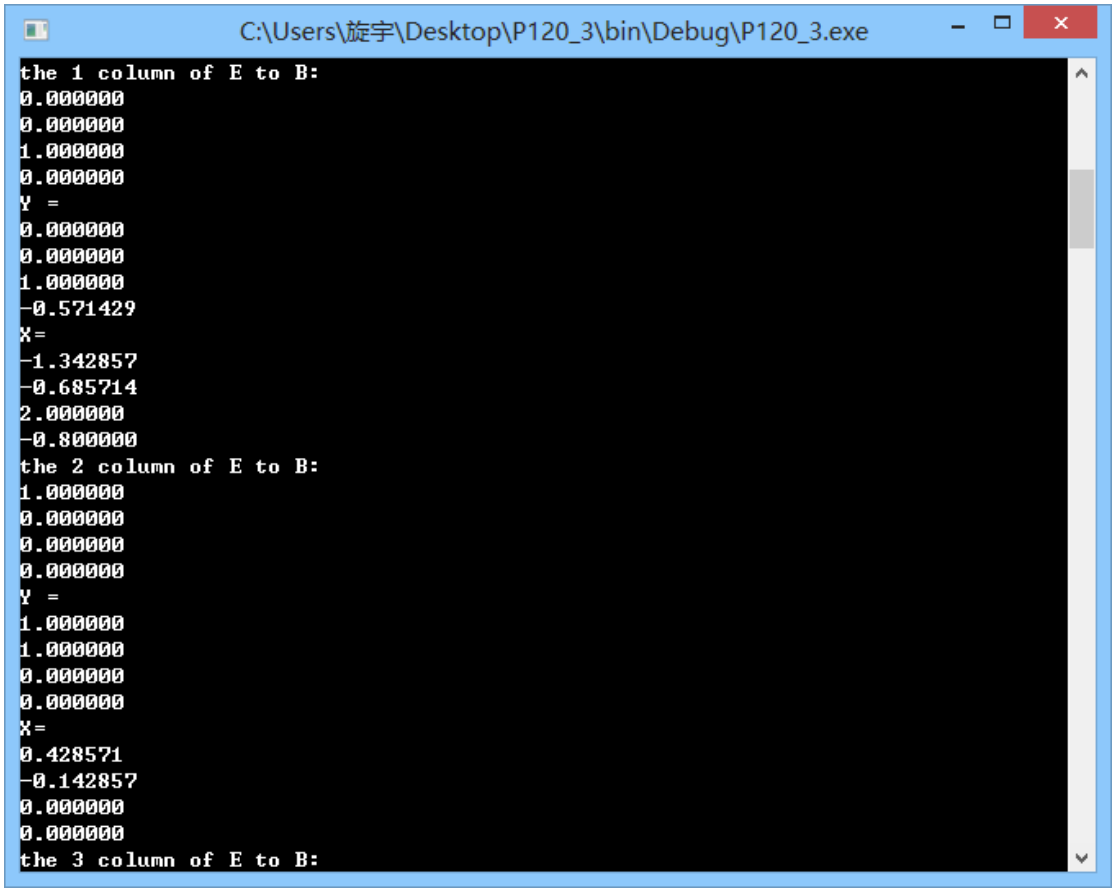

图 16.　P120 3.　$AC_1 = E_1$，$AC_2 = E_2$计算结果

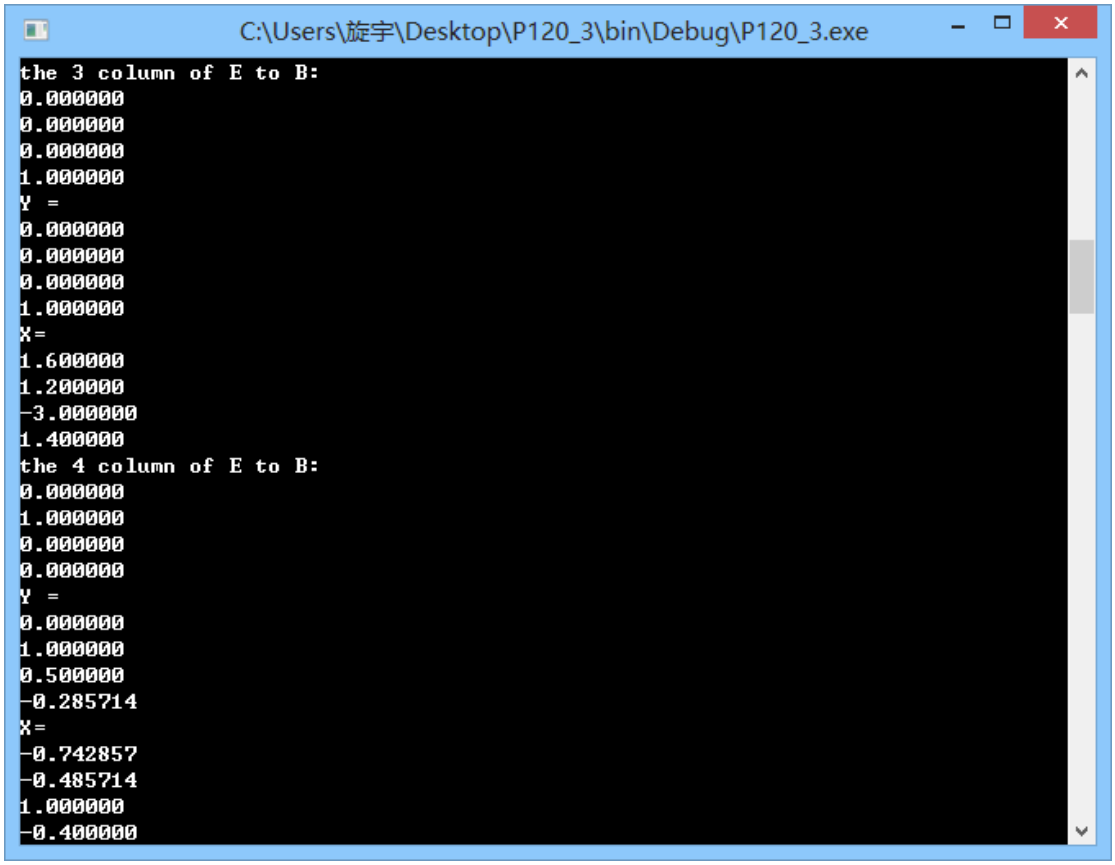

图 17.　P120 3.　$AC_3 = E_4$，$AC_4 = E_4$计算结果

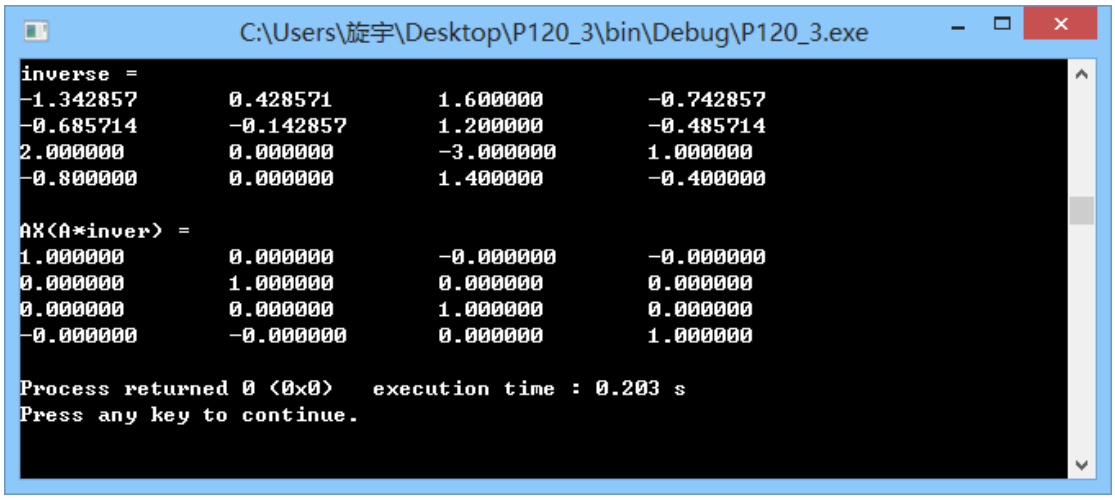

图 18.　P120 3.　A矩阵的逆的计算结果及其验证

5.　P129 3.

	对于方程组（a）解为 X =	对于方程组（b）解为 X =
1	0.633975	0.133975
2	0.464102	0.464102
3	0.509619	0.009619
4	0.497423	0.497423
5	0.500691	0.000691

6	0.499815	0.499815
7	0.500050	0.000050
8	0.499987	0.499987
9	0.500004	0.000004
10	0.499999	0.499999
11	0.500000	0.000000
12	0.500000	0.500000
13	0.500000	0.000000
14	0.500000	0.500000
15	0.500000	0.000000
16	0.500000	0.500000
17	0.500000	0.000000
18	0.500000	0.500000
19	0.500000	0.000000
20	0.500000	0.500000
21	0.500000	0.000000
22	0.500000	0.500000
23	0.500000	0.000000
24	0.500000	0.500000
25	0.500000	0.000000
26	0.500000	0.500000
27	0.500000	0.000000
28	0.500000	0.500000
29	0.500000	0.000000
30	0.500000	0.500000
31	0.500000	0.000000
32	0.500000	0.500000
33	0.500000	0.000000
34	0.500000	0.500000
35	0.500000	0.000000
36	0.500000	0.500000
37	0.500000	0.000000
38	0.500000	0.500000
39	0.500000	0.000000
40	0.500000	0.500000
41	0.499999	0.000000
42	0.500004	0.500000
43	0.499987	0.000000
44	0.500050	0.500000
45	0.499815	0.000000
46	0.500691	0.500000
47	0.497423	0.000000
48	0.509619	0.500000

49	0.464102	0.000000
50	0.633975	0.500000

表 1.　两个方程组的解

6. P130 4.

	（误差限度为 0.000001 时）X =	验证结果（A*X）	（误差限度为 0.00001 时）X =	验证结果（A*X）
1	0.463796	5.000000	0.463796	5.000000
2	0.537285	5.000000	0.537285	5.000000
3	0.509023	5.000000	0.509023	5.000000
4	0.498222	5.000000	0.498222	5.000000
5	0.498942	5.000000	0.498942	5.000000
6	0.499985	5.000000	0.499985	5.000000
7	0.500089	5.000000	0.500089	5.000000
8	0.500015	5.000000	0.500015	5.000000
9	0.499995	5.000000	0.499995	5.000000
10	0.499998	5.000000	0.499998	5.000000
11	0.500000	5.000000	0.500000	5.000000
12	0.500000	5.000000	0.500000	5.000000
13	0.500000	5.000000	0.500000	5.000000
14	0.500000	5.000000	0.500000	5.000000
15	0.500000	5.000000	0.500000	5.000000
16	0.500000	5.000000	0.500000	5.000000
17	0.500000	5.000000	0.500000	5.000000
18	0.500000	5.000000	0.500000	5.000000
19	0.500000	5.000000	0.500000	5.000000
20	0.500000	5.000000	0.500000	5.000000
21	0.500000	5.000000	0.500000	5.000000
22	0.500000	5.000000	0.500000	5.000000
23	0.500000	5.000000	0.500000	5.000000
24	0.500000	5.000000	0.500000	5.000000
25	0.500000	5.000000	0.500000	5.000000
26	0.500000	5.000000	0.500000	5.000000
27	0.500000	5.000000	0.500000	5.000000
28	0.500000	5.000000	0.500000	5.000000
29	0.500000	5.000000	0.500000	5.000000
30	0.500000	5.000000	0.500000	5.000000
31	0.500000	5.000000	0.500000	5.000000
32	0.500000	5.000000	0.500000	5.000000
33	0.500000	5.000000	0.500000	5.000000
34	0.500000	5.000000	0.500000	5.000000
35	0.500000	5.000000	0.500000	5.000000
36	0.500000	5.000000	0.500000	5.000000
37	0.500000	5.000000	0.500000	5.000001

38	0.500000	5.000000	0.500000	4.999997
39	0.500000	5.000000	0.500001	5.000006
40	0.500000	5.000000	0.500000	4.999995
41	0.499998	5.000000	0.499998	5.000000
42	0.499995	5.000000	0.499995	5.000003
43	0.500015	5.000000	0.500015	4.999999
44	0.500089	5.000000	0.500089	4.999999
45	0.499985	5.000000	0.499985	5.000000
46	0.498942	5.000000	0.498942	5.000000
47	0.498222	5.000000	0.498222	5.000000
48	0.509023	5.000000	0.509023	5.000000
49	0.537285	5.000000	0.537285	5.000000
50	0.463796	5.000000	0.463796	5.000000

表 2.　不同误差限度方程组的解及其验证

实验结论及分析

一、三角分解法的确可以用来解方程组。同时，在对需要行变换的非奇异矩阵进行三角分解的时候，在求转置矩阵 P 的时候不必在每一次交换行之后进行置换变换，而得到的结果也不会变化。至于为什么是这样的还需要进一步探索证明。

二、对于 P120 2.题，我们可以看出 A 矩阵是严重病态的矩阵，当 N 为 3 时其∞-条件数为 104, 当 N 为 11 时其∞-条件数为 3.9237e+07，当 N 为 11 时其∞-条件数为 1.8758e+14，但是在用偏序选主元策略计算解得时候还是能得到精确的解。证明偏序选主元策略是极其有效的解方程组的方法。一般求解情况下还是用一下。实验内容中要求解释所得结果和精确结果的差，但是发现没有误差。

三、高斯-赛德尔迭代中，向量的 1-范数可以有效地衡量两个向量的距离，在本题迭代中，误差限度为 0.00001 时结果的精度就在 5 位，但是误差限速为 0.000001 时结果的精度就达到了小数点后 6 位。

四、对于一些特殊的矩阵，利用矩阵的特征可以有效地减小计算次数。

实验代码

说明：代码比较多。每一道题的代码都由 main.c 以及 allmatrix.h 两部分组成。allmatrix.h 包含了这次实验所需的所有矩阵操作，每一道小题的 allmatrix.h 文件都相同。所以 allmatrix.h 文件代码只在最后展示一次。

1. <allmatrix.h>

```
#ifndef ALLMATRIX_H_INCLUDED
#define ALLMATRIX_H_INCLUDED

/************************************************************
Module Name: allmatrix.h
Module Date: 10/22/2014
```

```
Module Auth: Xuanyu Wang

Description: The operation of matrix would be used in this experiment.
      Revision History: matrix.h
************************************************************/
/*----------------Includes---------------*/
//the head files that were included
//#include <stdio.h>
//#include <stdlib.h>

/*---------Structures and Typedefs-------*/
/*None
*/

/*--------------Defines-----------------*/
#define a_mn (i * n_column + j +1)//a(m,n) is the true serial number of the element
#define a_jk ((j+1)*n_column+i+k)//the i+(j-i+1) line, i+k column element. i.e.

/*----------extern variables-------------*/
//the variables that were defined in other modules
//None

/*-----External Function Prototypes------*/
//the functions that were implemented in other modules
//None

#include <stdio.h>
#include <stdlib.h>

double* InitMatrix(int n_row,int n_column,double *matrix, int special);
double* SetMatrix(int n_row, int n_column, double* matrix);
void PrintMatrix(int n_row,int n_column,double *matrix);
void SolveAXB(int n_row, int n_column, double *A, double *X, double *B);
void TridiagEliminate(int n_row,int n_column,double *matrix, double *B);
double* TridiagMatrix(int n_row,int n_column,double *matrix);
double* AB(int n_row, int n_column, int single, double *A, double *B);
void CopyToColumn(int n_row, int n_column, double *matrix, int j, double *B);
void CopyToMatrix(int n_row, int n_column, double *X, int j,double *matrix);
void    BackSub4Y(int n_row, int n_column, double *L, double *Y, double *B);
void BackSub4X(int n_row, int n_column, double *U, double *X, double *Y);
void LUFact(int n_row,int n_column,double *matrix, double *idmatrix);
double* InitIdMatrix(int n_row,int n_column,double *idmatrix);
void Pivoting(int n_row,int n_column,double *matrix,double *P);
void GSIter4tri(int n_row, int n_column, double *A, double *B, double *X);
```

```
void GSIter4all(int n_row, int n_column, double *A, double *B, double *X);
double* InitX(int n_row,int n_column,double *matrix);
double* QuintudiagMatrix(int n_row,int n_column,double *matrix);
double* InitBMatrix(int n_row,double *B);

/*********************************************************
Function Name: InitMatrix
Function Description: Produce a matrix which dimension had been input in main().
                                                And the matrix's member is all 0.
Inputs: Three argument are required. The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too. The third
                is the point of the matrix and it should be double.
Outputs: Return a point of the matrix.
Notes:The module is individual
*********************************************************/
double* InitMatrix(int n_row,int n_column,double *matrix,int special)
{
    if(special == 0){
        int i,j;//counting the row and column
        matrix = (double *)malloc((n_row*n_column)*sizeof(double));//allocate the space for the matrix
        double value = 0;//define the matrix's value
        for(i = 0;i < n_row;i++){//for row
            for(j = 0;j < n_column;j++){//for column
                *(matrix + i* n_column + j ) = value;//give a(i+1,j+i) a value
            }
        }
        return (matrix);//return a one-dimension double array
    }
    else{
        int i,j;//counting the row and column
        matrix = (double *)malloc((n_row*n_column)*sizeof(double));//allocate the space for the matrix
        //int count = 0;//initialize the matrix's value
        for(i = 0; i < n_row; i++){//for row
            for(j = 0; j < n_column;j++){//for column
                *(matrix + a_mn - 1 ) = pow((double)(i+1),(double)(j));//give a(i+1,j+i) a value
            }
        }
        return (matrix);
    }
```

```
}

/************************************************************
Function Name: PrintMatrix
Function Description: The function display the matrix on the screen. The member of
                                                    each row is equal to the number of column.
Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too. The third
                is the point of the matrix and it should be double.
Outputs: There are no argument need to be return.
Notes:The matrix should be initialized.
************************************************************/
void PrintMatrix(int n_row,int n_column,double *matrix)
{
    printf("\n");
    int i,j;//for counting

    //display every member of the matrix
    for(i = 0;i < n_row;i++){
        for(j = 0;j < n_column;j++){
            printf("%f     ",*(matrix + i * n_column + j));//attention:the format should adjust when you need
            }
        printf("\n");
    }
}

/************************************************************
Function Name: ChangeMatrix
Function Description: Produce a matrix which dimension had been input in main().
                                                    And the member's value is decided by user.
Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too. The third
                is the point of the matrix and it should be double.
Outputs: Return a point of the matrix.
Notes:The matrix should be initialized.
************************************************************/
double* SetMatrix(int n_row, int n_column, double* matrix)
{
    int i,j;//counting the row and column
```

```
        //matrix = (double *)malloc((n_row*n_column)*sizeof(double));//allocate the space for the matrix

        for(i = 0; i < n_row; i++){//for row
            printf("the %d line is waiting for your:\n", i + 1);
            for(j = 0; j < n_column; j++){//for column
                printf("the a(%d,%d) = ", i + 1, j + 1);
                scanf("%lf", (matrix + a_mn - 1 ));//give a(i+1,j+i) a value
            }
        }

        return (matrix);
    }

    /************************************************************
    Function Name: TridiagMatrix
    Function Description: Produce a matrix which is a tridiagonal matrix.
    Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too. The third
                is the point of the matrix and it should be double.
    Outputs: Return a point of the matrix.
    Notes: The matrix should be initialized.
    ************************************************************/
    double* TridiagMatrix(int n_row,int n_column,double *matrix)
    {
        int i;//i is for counting
        for(i = 0;i < n_row;i++){
            //for the first row, we need two coefficients
            if((i + 1) == 1){
                printf("first row:input the coefficient of the a(%d,%d) & a(%d,%d):\n",1,1,1,2);
                scanf("%lf %lf",matrix,(matrix+1));
            }
            //for the last row, we need two coefficients
            else if((i + 1) == n_row){
                printf("input the coefficient of the a(%d,%d) & a(%d,%d):\n",n_row,n_row-1,n_row,n_row);
                scanf("%lf %lf",matrix+n_row*n_column-2,matrix+n_row*n_column-1);
            }
            //for each row except first and last need three coefficient
            else{
                printf("input the coefficient of the a(%d,%d) & a(%d,%d) &
```

```
a(%d,%d):\n",i+1,i,i+1,i+1,i+1,i+2);
                    scanf("%lf %lf %lf",matrix+i*n_column+i-1,matrix+i*n_column+i,matrix+i*n_column+i+1);
                }
            }
            return (matrix);//return the point of one-dimension array
        }

        /************************************************************
        Function Name: QuintudiagMatrix
        Function Description: Produce a matrix which is a tridiagonal system.
        Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
                    The second is the number of the column and it should be positive int,too. The third
                    is the point of the matrix and it should be double.
        Outputs: Return a point of the matrix.
        Notes: This function is depended on InitMatrix. Before you use this function, you should
                    built a matrix.
        ************************************************************/
        double* QuintudiagMatrix(int n_row,int n_column,double *matrix)
        {
            int i;
            for(i = 0;i < n_row;i++){
                //printf("\ni+1=%d\n",i+1);
                if((i + 1) == 1){
                    //printf("first row:input the coefficient of the a(%d,%d) & a(%d,%d) & a(%d,%d):\n",1,1,1,2,1,3);
                    scanf("%lf %lf %lf",matrix,(matrix+1),(matrix + 2));
                    //printf("a(1,1)=%f\ta(1,2)=%f\n",*matrix,*(matrix+1));//for testing
                }
                else if((i + 1) == 2){
                    //printf("first row:input the coefficient of the a(%d,%d) & a(%d,%d) & a(%d,%d) & a(%d,%d):\n",2,1,2,2,2,3,2,4);
                    scanf("%lf %lf %lf %lf",matrix+n_column,(matrix+1+n_column),(matrix + 2+n_column),(matrix + 3+n_column));
                }
                else if((i + 1) == n_row-1){
                    //printf("input the coefficient of the a(%d,%d) & a(%d,%d) & a(%d,%d) & a(%d,%d):\n",n_row-1,n_row-3,n_row-1,n_row-2,n_row-1,n_row-1,n_row-1,n_row);
                    scanf("%lf %lf %lf %lf",matrix+(n_row-1)*n_column-4,matrix+(n_row-1)*n_column-3,matrix+(n_row-1)*n_column-2,matrix+(n_row-1)*n_column-1);
                }
                else if((i + 1) == n_row){
```

```
                    //printf("input the coefficient of the a(%d,%d) & a(%d,%d) & a(%d,%d):\n",n_row,n_row-2,n_row,n_row-1,n_row,n_row);
                    scanf("%lf %lf %lf",matrix+n_row*n_column-3,matrix+n_row*n_column-2,matrix+n_row*n_column-1);
                    //printf("a(%d,%d)=%f\ta(%d,%d)=%f",n_row,n_row-1,*(matrix+n_row*n_column-2),n_row,n_row,*(matrix+n_row*n_column-1));//for testing
                }
                else{
                    //printf("input the coefficient of the a(%d,%d) & a(%d,%d) & a(%d,%d):\n",i+1,i,i+1,i+1,i+1,i+2);
                    scanf("%lf %lf %lf %lf %lf",matrix+i*n_column+i-2,matrix+i*n_column+i-1,matrix+i*n_column+i,matrix+i*n_column+i+1,matrix+i*n_column+i+2);
                    //printf("a(%d,%d)=%f & a(%d,%d)=%f & a(%d,%d)=%f\n",i+1,i,*(matrix+i*n_row+i-1),i+1,i+1,*(matrix+i*n_row+i),i+1,i+2,*(matrix+i*n_row+i+1));
                }
            }
            return matrix;
        }

        /************************************************************
        Function Name: TridiagEliminate
        Function Description: Use the Frobenius matrix to eliminate the tridiagonal matrix.
        Inputs: The n_row is the number of "matrix"'s rows. It's a positive int.
                    The n_column is the number of the column and it's positive int,too.
                    matrix plus B to produce a argumented matrix. The B are n_row*1 matrixs.
        Outputs: None
        Notes:The matrix should be initialized and is a tridiagonal matrix.
        ************************************************************/
        void TridiagEliminate(int n_row,int n_column,double *matrix, double *B)
        {
            int i = 0;
            double coe;
            for(i = 0;i < n_row - 1 ;i++){
                    //calculate the coefficient to eliminate the next row
                    coe = -(*(matrix + i * n_column + i + n_column) / (*(matrix + i * n_column + i)) );
                   // printf("coe = %lf\n",coe);//for testing
                   //the operation of eliminate. The B need one step, and the matrix need two steps
                    *(matrix + i * n_column + i + n_column) = 0;
                    *(B + i * 1 + 1) = *(B + i*1)*coe+*(B + i * 1 + 1);
                    *(matrix + i * n_column + i + n_column + 1) = *(matrix + i * n_column + i + 1) * coe + (*(matrix + i * n_column + i + n_column + 1));
```

```
        }
    }

    /*************************************************************
    Function Name: SolveAXB
    Function Description: Solve the matrix A*X=B.
    Inputs: The n_row is the number of A's rows. It's a positive int.
                The n_column is the number of the column and it's positive int,too. The third
                is the point of the A. The X & B are n_row*1 matrixs.
    Outputs: Display the matrix of the X(solution).
    Notes:The function need PrintMatrix function. And in this project,
                A and B had been eliminated.
    *************************************************************/
    void SolveAXB(int n_row, int n_column, double *A, double *X, double *B)
    {
        int i = 0, j = 0;//for counting
        //'sum' is the sum of    i line of A timed with X, then reduce the a(i,i)*X(i)
        double sum = 0;
        //from the last row to solve the equation
        for(i = n_row - 1; i >= 0; i--){
            //for the last row we solve the unknown x_n directly
            if(i == n_row - 1){
                //display the operation for debug
                //x_n = b_n / d_n
                //printf("\ni = %d, %lf = %lf / %lf", i, *(B + i) / *(A + i*n_column + i), *(B + i), *(A + i*n_column + i));
                *(X + i) = *(B + i) / *(A + i*n_column + i);
            }
            //for the other rows we need the [B(i)-sum]/a(i,i) to calculate the x(n)
            else{
                sum = *(X + i + 1) * (*(A + i*n_column + i + 1));
                //display the operation to debug
                //printf("\nsum = %lf",sum);
                //printf("\ni = %d, %lf = (%lf - %lf) / %lf", i, (*(B + i) -    sum)/ *(A + i*n_column + i), *(B + i),sum, *(A + i*n_column + i));
                *(X + i) = (*(B + i) -    sum)/ *(A + i*n_column + i);
            }
            sum = 0;
        }
        //display the solution of A*X = B
        printf("\nX= ");
        PrintMatrix(n_row, 1, X);
    }
```

```
/************************************************************
Function Name: AB
Function Description: to calculate the result of A times B
Inputs:    The first is the number of A's row. It should be a positive int.
                The second is the number of the A's column and it should be positive int,too. The third
                is the number of the B's column. The forth is matrix A and the fifth is matrix B
Outputs: Return the result of A*B.
Notes:The function assume A's column is equal with B's row.
************************************************************/
double* AB(int n_row, int n_column, int single, double *A, double *B)
{
    printf("\n");
    double *result;//the cache to save the A*B
    result = InitIdMatrix(n_row,n_column,result);//initialize the result
    int i,j,k;//for counting
    double sum = 0;//the sum of a(i,)*b(,j)
    for(i = 0; i < n_row; i++){
        for(j = 0; j < single; j++){
            //calculate the value of ab(i,j)
            for(k = 0; k < n_column; k++){
                sum += (*(A + i*n_row + k)) * (*(B + k*single + j));
            }
            *(result + i*single + j) = sum;//assign the value to ab(i,j)
            //printf("result = %lf\n\n",*(result + i*single + j));
            sum = 0;//reset the sum for next calculating
        }

    }
    return result;
}

/************************************************************
Function Name: CopyToColumn
Function Description: copy one column of a matrix to a vector.
Inputs:The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too.
                the matrix is a matrix. the j is matrix's j column will be copied to B.
Outputs: There are no argument need to be return.
Notes:
************************************************************/
void CopyToColumn(int n_row, int n_column, double *matrix, int j, double *B)
```

```
{
	int i;
	//for each row, find the jth number.
	for(i = 0; i < n_row; i++){
		*(B+i) = *(matrix + i*n_row + j);
	}
}

/************************************************************
Function Name: CopyToMatrix
Function Description: copy one column of a matrix to a vector.
Inputs:The first is the number of row. It should be a positive int.
		The second is the number of the column and it should be positive int,too.
		the matrix is a matrix. the j is matrix's j column will be assigned from B.
Outputs: There are no argument need to be return.
Notes:
************************************************************/
void CopyToMatrix(int n_row, int n_column, double *X, int j,double *matrix)
{
	int i;
	//for each row, change the jth element.
	for(i = 0; i < n_row; i++){
		*(matrix + i*n_row + j) = *(X+i);
	}
}

/************************************************************
Function Name: BackSub4Y
Function Description: solve the Y in LY=B.
Inputs:The first is the number of row. It should be a positive int.
		The second is the number of the column and it should be positive int,too.
Outputs: There are no argument need to be return.
Notes:
************************************************************/
void	BackSub4Y(int n_row, int n_column, double *L, double *Y, double *B)
{
	int i = 0, j = 0;
	double sum_0i = 0;//the sum of l(i,j)*y(j), j is from 0 to i.
	for(i = 0; i < n_column; i++){
		//get the sum
		for(j = 0; j < i; j++){
			sum_0i += *(L + a_mn - 1) * (*(Y+ j));
		}
	//solve the Y(i)
```

```
            *(Y + i) = (*(B + i) - sum_0i) / *(L + i*n_column + i);//calculate the solution of
Y_i
            sum_0i = 0;//reset the sum for next calculation
        }
        printf("Y = ");
        PrintMatrix(n_row, 1, Y);
    }

    /************************************************************
    Function Name: BackSub4X
    Function Description: solve the X in UX=Y.
    Inputs:The first is the number of row. It should be a positive int.
                The second is the number of the column and it should be positive int,too.
    Outputs: There are no argument need to be return.
    Notes:
    ************************************************************/
    void BackSub4X(int n_row, int n_column, double *U, double *X, double *Y)
    {
        int i = 0, j = 0;
        double sum_iend = 0;//the sum of u(i,j)*x(j), j is from end to i.
        for(i = n_row - 1; i >= 0; i--){
            //calculate the sum
            for(j = n_column - 1; j > i; j--){
                sum_iend += *(U + a_mn - 1) * (*(X+ j));
            }
        //calculate the X(i)
            *(X + i) = (*(Y + i) - sum_iend) / *(U + i*n_column + i);//calculate the solution
of Y_i
            sum_iend = 0;//reset the sum for next calculation
        }
        printf("X= ");
        PrintMatrix(n_row, 1, X);
    }

    /************************************************************
    Function Name: InitIdMatrix
    Function Description: Produce a matrix which dimension is n_row*n_column
                                                and the diagonal element is 1 and
other elements are 0.
    Inputs: Three argument are asked. The first is the number of row. It should be a positive
int.
            The second is the number of the column and it should be positive int,too.
The third
            is the point of the matrix and it should be double.
```

```
Outputs: Return the point of the matrix.
Notes:
************************************************************/
double* InitIdMatrix(int n_row,int n_column,double *idmatrix)
{
    int i,j;//counting the row and column
    idmatrix = (double *)malloc((n_row*n_column)*sizeof(double));//allocate the space for the matrix
    int ii = 1;//when the element in diagonal line, assign the element ii
    int ij = 0;//when the element isn't in diagonal line, assign the element ij
    for(i = 0; i < n_row; i++){//for row
        for(j = 0; j < n_column;j++){//for column
            if(i+1 == j+1){
                *(idmatrix + a_mn - 1 ) = ii;//give a(i+1,j+i) a value
            }
            else{
                *(idmatrix + a_mn - 1 ) = ij;
            }
        }
    }
    return (idmatrix);
}

/************************************************************
Function Name: LUFact
Function Description: The function to find the L and U with triangular factorization
Inputs:The first is the number of row. It should be a positive int.
        The second is the number of the column and it should be positive int,too. The third
        is the point of the matrix and it should be double and it will be U. The idmatrix is a
        identical matrix.
Outputs: There are no argument need to be return.
Notes:
************************************************************/
void LUFact(int n_row,int n_column,double *matrix, double *idmatrix)
{
    int i = 0, j = 0,k = 0;
    double coe;//the coefficient to eliminate
    for(i = 0;i < n_row - 1 ;i++){
        //if the matrix is a 1*1 dimension matrix. There are no need to be factorized.
        if(n_row == 1){
            return;
        }
```

```
                //if the matrix need to be factorized.
                for(j = i; j < n_row - 1; j++){
                        //find the coefficient for interchange
                    coe = -(*(matrix    + i + (j+1)*n_column) / (*(matrix + i * n_column + i)) );
                    //printf("coe = %lf\n",coe);
                    for(k = 0; k + i < n_column; k++){
                        //interchange the matrix and identical matrix to get the U and P
                        *(matrix + a_jk) = *(matrix + i * n_column +i + k) * coe + (*(matrix +
a_jk));
                        *(idmatrix + a_jk) = *(idmatrix + i * n_column + i + k) * (-coe) +
(*(idmatrix    + a_jk));
                    }
                }
            }
        }


        /************************************************************
        Function Name: Pivoting
        Function Description: To find the interchange matrix of matrix with using partial pivoting.
        Inputs: The n_row is the number of "matrix"'s rows. It's a positive int.
                    The n_column is the number of the column and it's positive int,too.
                    the matrix is which would be pivoted. The P is the interchange matrix.
        Outputs: None
        Notes:
        ************************************************************/
        void Pivoting(int n_row,int n_column,double *matrix,double *P)
        {
            double maxium = 0;
            double *temp;//save the row would be changed
            temp = (double*)malloc(n_column*sizeof(double));
            int i, j;//for counting
            int mark;//save the line which will interchange with current line.
            int fc;//just for interchange two lines of a matrix

            for(i = 0; i < n_column; i++){
        //*********************************************************************
*****************
                    //for each row,you'll find the most max element of i column
                    //in this section, the max will be founded and the line will be marked
                    maxium = *(matrix + i*n_column + i);
                    mark = i;
                    for(j = i+1; j < n_row; j++){//from i line to row-1 line
                        if(fabs(maxium) < fabs(*(matrix+j*n_row+i))){//if the max is samller
```

```
than a(j,i)
                                //printf("j = %d\t maxium = %lf\t*(matrix+j*n_row+i)
= %lf\n",j,maxium,*(matrix+j*n_row+i));
                                maxium = *(matrix+j*n_row+i);//assign the max from a(j,i)
                                mark = j;//the j line will be marked
                        }
                }
                //printf("\nmax = %lf\n",maxium);
                //printf("\nthe %d line need to interchange with %d line\n",i+1,mark+1);
                //there are need to interchange with itself.
                if(mark == i){
                        continue;
                }
        //**********************************************************************
                //then, you need interchange the i line with "mark" line
                for(fc = 0; fc < n_column; fc++){
                        *(temp+fc) = *(matrix+mark*n_row+fc);//put a(mk,fc) in cache t(fc)
                        *(matrix+mark*n_row+fc) = *(matrix+i*n_row+fc);//a(mk,fc) = a(i,fc)
                        *(matrix+i*n_row+fc) = *(temp +fc);//a(i,fc)=t(fc)
                }
                //To make the P.
                for(fc = 0; fc < n_column; fc++){
                        *(temp+fc) = *(P+mark*n_row+fc);//put a(mk,fc) in cache t(fc)
                        *(P+mark*n_row+fc) = *(P+i*n_row+fc);//a(mk,fc) = a(i,fc)
                        *(P+i*n_row+fc) = *(temp +fc);//a(i,fc)=t(fc)
                }
                //PrintMatrix(n_row,n_column,matrix);
        //**********************************************************************
              }

        }

        /************************************************************
        Function Name: GSIter
        Function Description: Use the Gauss-Seidel iteration to solve the matrix euqation.
        Inputs: Three argument are asked. The first is the number of row. It should be a positive
int.
                The second is the number of the column and it should be positive int,too.
The third
                is the point of the matrix and it should be double. Generally, the matrix is
a
                n*n matrix and the X and B are n*1 matrix.
        Outputs: Print the solution X.
        Notes:
```

```
**********************************************************/
void GSIter4tri(int n_row, int n_column, double *A, double *B, double *X)
{
	int i,j,k,single = 1;
	double sum,tol = 0.000000001,err;
	double *te;//cache for the older solution
	double a[5]={0};
	PrintMatrix(n_row,n_column,A);
	te = InitX(n_row,single,te);
	//the circle will be ended while the error<tolerance
	do{
		for(k = 0;k < n_column;k++){
			*(te+k) = *(X+k);
		}
		//calculate the sum
		for(i = 0; i < n_row; i++){
			for(j = i-1; j < i+2; j++){
				if((j>=0)&&(j<n_column)&&(j != i) ){
					sum += *(X + j) * (*(A + i*n_column + j));
				}
			}
		//calculate the X(i)
			*(X + i) = ((*(B + i)) - sum) / (*(A + i*n_column + i));
			sum = 0;
		}
		err = 0;
		//calculate the error
		for(k = 0;k < n_column;k++){
			err += fabs(*(X+k)-*(te+k));
		}

	}while(err > tol);
	printf("X = ");
	PrintMatrix(n_row, 1, X);
}

/************************************************************
Function Name: GSIter
Function Description: Use the Gauss-Seidel iteration to solve the matrix euqation.
Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
		The second is the number of the column and it should be positive int,too. The third
		is the point of the matrix and it should be double. Generally, the matrix is
```

a
n*n matrix and the X and B are n*1 matrix.
Outputs: Print the solution X.
Notes:
**********/
```
void GSIter4all(int n_row, int n_column, double *A, double *B, double *X)
{
    int i,j,k,single = 1;
    double sum,tol = 0.00001,err;
    double *te;
    double a[5]={0};
    PrintMatrix(n_row,n_column,A);
    te = InitX(n_row,single,te);
    do{
        for(k = 0;k < n_column;k++){
            *(te+k) = *(X+k);
        }
        for(i = 0; i < n_row; i++){
            for(j = 0; j < n_column; j++){
                if(j != i){
                    sum += *(X + j) * (*(A + i*n_column + j));
                }
            }
            *(X + i) = ((*(B + i)) - sum) / (*(A + i*n_column + i));
            sum = 0;
        }
        err = 0;
        for(k = 0;k < n_column;k++){
            err += fabs(*(X+k)-*(te+k));
        }

    }while(err > tol);
    printf("X = ");
    PrintMatrix(n_row, 1, X);
}

/************************************************************
Function Name: InitX
Function Description: Produce a matrix which dimension had been input in main().
                      And the matrix's member is all 0.
Inputs: Three argument are asked. The first is the number of row. It should be a positive int.
        The second is the number of the column and it should be positive int,too. The third
```

```
                    is the point of the matrix and it should be double.
    Outputs: Return a point of the matrix.
    Notes:
    ************************************************************/
    double* InitX(int n_row,int n_column,double *matrix)
    {
        int i,j;//counting the row and column
        matrix = (double *)malloc((n_row*n_column)*sizeof(double));//allocate the space
for the matrix
        double coun = 0.49;//initialize the matrix's value
        for(i = 0;i < n_row;i++){//for row
            for(j = 0;j < n_column;j++){//for column
                *(matrix + i* n_column + j ) = coun;//give a(i+1,j+i) a value
            }
        }
        return (matrix);
    }

    /************************************************************
    Function Name: InitBMatrix
    Function Description: Produce a vector B.
    Inputs: The first is the number of row. It should be a positive int.
    Outputs: Return the point of B.
    Notes:
    ************************************************************/
    double* InitBMatrix(int n_row,double *B)
    {
        int i,j;//counting the row and column
        int n_column=1;
        B = (double *)malloc(n_row*sizeof(double));//allocate the space for the matrix
        //int count = 0;//initialize the matrix's value
        *(B+0) = (double)(n_row);
        for(i = 1; i < n_row; i++){//for row
                *(B + i ) = (pow((double)(i+1), n_row) - 1) / (i);//give a(i+1,1) a value
        }
        return (B);
    }
#endif // ALLMATRIX_H_INCLUDED
```

2. P108 1.

<main.c>

```
    #include <stdio.h>
    #include <stdlib.h>
    #include "allmatrix.h"
    int main()
```

```
{
    freopen("in2.txt","r",stdin);//for testing, load data from a file
    double *matrix, *X, *B;//matrix*X = B;matrix is the tridiagonal matrix.
    int n_row,n_column;//the number of matrix's row and column
    int single = 1;//the number of X and B's column

    //printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
    scanf("%d%d",&n_row,&n_column);//Get the dimension of the matrix

    matrix = InitMatrix(n_row,n_column,matrix,0);//produce a matrix, all elements are
0
    X = InitMatrix(n_row, single, X,0);//produce a X, all elements are 0
    B = InitMatrix(n_row, single, B,0);//produce a B, all elements are 0

    matrix = TridiagMatrix(n_row,n_column,matrix);//change the matrix into a
tridiagonal matrix
    B = SetMatrix(n_row, 1, B);//set the B
    //display the matrix
    printf("\nTriMatrix = ");
    PrintMatrix(n_row,n_column,matrix);//display the matrix
    //display the B
    printf("\nB = ");
    PrintMatrix(n_row, 1, B);//display the matrix

    //eliminate the matrix's elements
    TridiagEliminate(n_row,n_column,matrix,B);
    //display the matrix after elimination
    printf("\nAfter the elimination the matrix = ");
    PrintMatrix(n_row,n_column,matrix);
    //display the matrix after elimination with matrix
    printf("\nAfter the elimination the B = ");
    PrintMatrix(n_row,1,B);

    //solve the equation matrix*X = B
    SolveAXB(n_row, n_column, matrix, X, B);
}
```

3. P120 1.

<main.c>

```
//Notes:All the Printf() or PrintMatrix() which had been hidden are for testing
#include <stdio.h>
#include <stdlib.h>
#include "allmatrix.h"
int main()
{
```

```
    freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    //matrix*X = B;matrix2=matrix. inver is the inverse of matrix.P*matrix=LU.
    double *matrix, *matrix2, *U, *P, *inver;
    //E & idmatrix are identity matrix.
    double *E, *L, *idmatrix;
    //UX=Y,LY=B
    double *B, *X, *Y;
    int n_row,n_column;//the dimension of the matrix.
    int j;//for counting

    //To get the dimension of the matrix
    printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
    scanf("%d %d",&n_row,&n_column);

/*--intialize the matrix--*/
    matrix = InitMatrix(n_row, n_column, matrix,0);
    matrix2 = InitMatrix(n_row, n_column, matrix2,0);
    //inver = InitMatrix(n_row, n_column, inver);
    //E = InitIdMatrix(n_row, n_column, E);
    P = InitIdMatrix(n_row, n_column, P);
    idmatrix = InitIdMatrix(n_row, n_column, idmatrix);

/*--assign the value of each matrix's element--*/
    matrix = SetMatrix(n_row, n_column, matrix);
    matrix2 = SetMatrix(n_row, n_column, matrix2);
/*--show the matrixs--*/
    printf("\nthe A(matrix) = ");
    PrintMatrix(n_row, n_column, matrix);
    //PrintMatrix(n_row, n_column, E);

/*--To intialize the matrix, and assign the value of each matrix's element--*/
    B = InitMatrix(n_row, 1, B,0);
    B = SetMatrix(n_row, 1, B);
    printf("\nB = ");
    PrintMatrix(n_row, 1, B);

/*--pivote the matrix--*/
    Pivoting(n_row, n_column, matrix,P);
    //change the B accordingly
    B = AB(n_row,n_column,1,P,B);
    printf("after pivoting, P = ");
    PrintMatrix(n_row,n_column,P);
    printf("after pivoting, B = ");
```

```
        PrintMatrix(n_row,1,B);
        printf("after pivoting, matrix = ");
        PrintMatrix(n_row,n_column,matrix);
/*--Using the triangular factorization to factorize the matrix--*/
//matrix had been changed
        LUFact(n_row, n_column, matrix, idmatrix);

/*--show the result of the matrix had been factorized--*/
        printf("L = ");//show the lower-triangular matrix
        L = idmatrix;
        PrintMatrix(n_row, n_column, L);
        printf("U = ");//show the upper-triangular matrix
        U = matrix;
        PrintMatrix(n_row, n_column, U);

        X = InitMatrix(n_row, 1, X,0);
        Y = InitMatrix(n_row, 1, Y,0);

        //printf("check the B will use:");
        //PrintMatrix(n_row,1,B);
        /*for(j = 0; j < n_column; j++){
                CopyToColumn(n_row,n_column,E,j,B);
                printf("the %d column of E to B:",j+1);
                PrintMatrix(n_row, 1,B);
                BackSub4Y(n_row, n_column, L, Y, B);
                BackSub4X(n_row, n_column, U, X, Y);
                CopyToMatrix(n_row,n_column,X,j,inver);
        }
        printf("inverse = ");
        PrintMatrix(n_row,n_column,inver);*/

        /*printf("X = ");
        PrintMatrix(n_row, 1, X);
        B=AB(n_row,n_column,1,U,X);
        printf("UX = ");
        PrintMatrix(n_row, 1, B);
        B=AB(n_row,n_column,1,L,B);
        printf("LUX = ");
        PrintMatrix(n_row, 1, B);
        printf("A=");
        PrintMatrix(n_row,n_column,matrix2);*/
        /*--use back sub to calculate X and Y--*/
        BackSub4Y(n_row, n_column, L, Y, B);
        BackSub4X(n_row, n_column, U, X, Y);
```

```
        printf("AX(A*X) = ");
        B = AB(n_row,n_column,1,matrix2,X);
        PrintMatrix(n_row,1,B);
        return 0;
}
```

4. P120 2.

<main.c>

```
#include <stdio.h>
#include <stdlib.h>
#include "allmatrix.h"
int main()
{
    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    double *matrix, *U, *P;
    double *idmatrix, *L;
    double *B, *X, *Y;
    int n_row,n_column;
    int i;
    printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
    scanf("%d %d",&n_row,&n_column);

/*--intialize the matrix which would be triangular factorize and the identity matrix--*/
    matrix = InitMatrix(n_row, n_column, matrix,1);
    double *matrix2;
    matrix2 = InitMatrix(n_row, n_column, matrix2,1);
    idmatrix = InitIdMatrix(n_row, n_column, idmatrix);
    P = InitIdMatrix(n_row, n_column, P);

/*--show the matrixs--*/
    printf("\nthe A(matrix) = ");
    PrintMatrix(n_row, n_column, matrix);
    //PrintMatrix(n_row, n_column, idmatrix);

    B = InitBMatrix(n_row, B);
    printf("\nB = ");
    PrintMatrix(n_row, 1, B);
//pivoting the matrix
    Pivoting(n_row, n_column, matrix,P);
    B = AB(n_row,n_column,1,P,B);
    printf("after pivoting, P = ");
    PrintMatrix(n_row,n_column,P);
    printf("after pivoting, B = ");
```

```
        PrintMatrix(n_row,1,B);
        printf("after pivoting, matrix = ");
        PrintMatrix(n_row,n_column,matrix);
    /*--Using the triangular factorization to factorize the matrix--*/
    //matrix had been changed
        LUFact(n_row, n_column, matrix, idmatrix);

    /*--show the result of the matrix had been factorized--*/
        printf("L = ");//show the lower-triangular matrix
        L = idmatrix;
        PrintMatrix(n_row, n_column, L);
        printf("U = ");//show the upper-triangular matrix
        U = matrix;
        PrintMatrix(n_row, n_column, U);

        X = InitMatrix(n_row, 1, X,0);
        Y = InitMatrix(n_row, 1, Y,0);
        //printf("check the B will used:");
        //PrintMatrix(n_row,1,B);
        /*--use back sub to calculate X and Y--*/
        BackSub4Y(n_row, n_column, L, Y, B);
        BackSub4X(n_row, n_column, U, X, Y);

        /*printf("X = ");
        PrintMatrix(n_row, 1, X);
        B=AB(n_row,n_column,1,U,X);
        printf("UX = ");
        PrintMatrix(n_row, 1, B);
        B=AB(n_row,n_column,1,L,B);
        printf("LUX = ");
        PrintMatrix(n_row, 1, B);
        printf("A=");
        PrintMatrix(n_row,n_column,matrix2);*/
        //verify the answer
        B = AB(n_row,n_column,1,matrix2,X);
        printf("AX = ");
        PrintMatrix(n_row,1,B);
        return 0;
}
```

5. P120 3.

<main.c>

```
#include <stdio.h>
#include <stdlib.h>
#include "allmatrix.h"
```

```
int main()
{
	freopen("in.txt","r",stdin);
	//freopen("out.txt","w",stdout);
	double *matrix, *matrix2, *U, *P, *inver;
	double *E, *L, *idmatrix;
	double *B, *X, *Y;
	int n_row,n_column;
	int j;
	printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
	scanf("%d %d",&n_row,&n_column);

/*--intialize the matrix which would be triangular factorize and the identity matrix--*/
	matrix = InitMatrix(n_row, n_column, matrix,0);
	matrix2 = InitMatrix(n_row, n_column, matrix2,0);
	inver = InitMatrix(n_row, n_column, inver,0);
	E = InitIdMatrix(n_row, n_column, E);
	P = InitIdMatrix(n_row, n_column, P);
	idmatrix = InitIdMatrix(n_row, n_column, idmatrix);

/*--assign the value of each matrix's element--*/
	matrix = SetMatrix(n_row, n_column, matrix);
	matrix2 = SetMatrix(n_row, n_column, matrix2);
/*--show the matrixs--*/
	printf("\nthe A(matrix) = ");
	PrintMatrix(n_row, n_column, matrix);
	//PrintMatrix(n_row, n_column, E);

	B = InitMatrix(n_row, 1, B,0);
	//B = SetMatrix(n_row, 1, B);
	//printf("\nB = ");
	//PrintMatrix(n_row, 1, B);

//pivoting the matrix
	Pivoting(n_row, n_column, matrix,P);
	E = AB(n_row,n_column,n_column,P,E);
	printf("!!!after pivoting, P = ");
	PrintMatrix(n_row,n_column,P);
	printf("after pivoting, E = ");
	PrintMatrix(n_row,n_column,E);
	printf("after pivoting, matrix = ");
	PrintMatrix(n_row,n_column,matrix);
/*--Using the triangular factorization to factorize the matrix--*/
```

```
//matrix had been changed
	LUFact(n_row, n_column, matrix, idmatrix);

/*--show the result of the matrix had been factorized--*/
	printf("L = ");//show the lower-triangular matrix
	L = idmatrix;
	PrintMatrix(n_row, n_column, L);
	printf("U = ");//show the upper-triangular matrix
	U = matrix;
	PrintMatrix(n_row, n_column, U);

	X = InitMatrix(n_row, 1, X,0);
	Y = InitMatrix(n_row, 1, Y,0);

	//printf("check the B will used:");
	//PrintMatrix(n_row,1,B);
	//for each column,calculate the solution of it
	for(j = 0; j < n_column; j++){
		//get a column
		CopyToColumn(n_row,n_column,E,j,B);
		printf("the %d column of E to B:",j+1);
		PrintMatrix(n_row, 1,B);
	//solve it
		BackSub4Y(n_row, n_column, L, Y, B);
		BackSub4X(n_row, n_column, U, X, Y);
		CopyToMatrix(n_row,n_column,X,j,inver);
	}
	//display the result of [C]
	printf("inverse = ");
	PrintMatrix(n_row,n_column,inver);

	/*printf("X = ");
	PrintMatrix(n_row, 1, X);
	B=AB(n_row,n_column,1,U,X);
	printf("UX = ");
	PrintMatrix(n_row, 1, B);
	B=AB(n_row,n_column,1,L,B);
	printf("LUX = ");
	PrintMatrix(n_row, 1, B);
	printf("A=");
	PrintMatrix(n_row,n_column,matrix2);*/
	E = AB(n_row,n_column,n_column,matrix2,inver);
	//verify the result
	printf("AX(A*inver) = ");
```

```
        PrintMatrix(n_row,n_column,E);
        return 0;
}
```

6. P129 3.

<main.c>

```
#include <stdio.h>
#include <stdlib.h>
#include "allmatrix.h"

int main()
{
    freopen("in1.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    double *matrix, *X, *B ,*result;
    int n_row,n_column;
    int single = 1;

    printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
    scanf("%d%d",&n_row,&n_column);//Get the dimension of the matrix

    matrix = InitMatrix(n_row,n_column,matrix,0);//produce a matrix
    X = InitX(n_row, single, X);
    B = InitMatrix(n_row, single, B,0);
    result = InitMatrix(n_row,single,result,0);
    //printf("Matrix = ");
    //PrintMatrix(n_row,n_column,matrix);//display the matrix
    //printf("\nX = ");
    //PrintMatrix(n_row, single, X);
    //printf("\nB = ");
    //PrintMatrix(n_row, single, B);*/
    //printf("\nresult = ");
    //PrintMatrix(n_row, single, result);

    matrix = TridiagMatrix(n_row,n_column,matrix);//change the matrix into a tridiagonal matrix
    B = SetMatrix(n_row, 1, B);
    printf("\nTriMatrix = ");
    PrintMatrix(n_row,n_column,matrix);//display the matrix
    printf("\nB = ");
    PrintMatrix(n_row, 1, B);//display the matrix

    GSIter(n_row, n_column,matrix, B, X);
    result = AB(n_row,n_column,single,matrix,X);
    PrintMatrix(n_row,single,result);
```

```
}
```

7. P129 4.

<main.c>

```
#include <stdio.h>
#include <stdlib.h>
#include "allmatrix.h"
int main()
{
    freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);
    double *matrix, *X, *B ,*result;
    int n_row,n_column;
    int single = 1;

    printf("please input the dimonsion of the Matrix you'll make:m & n:\n");
    scanf("%d%d",&n_row,&n_column);//Get the dimension of the matrix
/*--To intialize the matrix, and assign the value of each matrix's element--*/
    matrix = InitMatrix(n_row,n_column,matrix,0);//produce a matrix
    X = InitX(n_row, single, X);
    B = InitMatrix(n_row, single, B,0);
    result = InitMatrix(n_row,single,result,0);
    //printf("Matrix = ");
    //PrintMatrix(n_row,n_column,matrix);//display the matrix
    //printf("\nX = ");
    //PrintMatrix(n_row, single, X);
    //printf("\nB = ");
    //PrintMatrix(n_row, single, B);*/
    //printf("\nresult = ");
    //PrintMatrix(n_row, single, result);
/*--To intialize the matrix, and assign the value of each matrix's element--*/
    matrix = QuintudiagMatrix(n_row,n_column,matrix);//change the matrix into a tridiagonal matrix
    B = SetMatrix(n_row, 1, B);
    printf("\nTriMatrix = ");
    PrintMatrix(n_row,n_column,matrix);//display the matrix
    printf("\nB = ");
    PrintMatrix(n_row, 1, B);//display the matrix

/*--use Gauss-seidal iteration to solve--*/
    GSIter4all(n_row, n_column,matrix, B, X);
    result = AB(n_row,n_column,single,matrix,X);
    PrintMatrix(n_row,single,result);
}
```