Problem Set 6 **R08244002 Shin-Rong, Tsai**

- ## LU-Decomposition with Pivoting

1. Result

Run the code *LUDecompose_Pivote.c*.

```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ gcc -o b.out LUDecompose_Pivote.c

cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ ./b.out
A =
1.00000e+00 1.00000e+00 0.00000e+00
1.00000e+00 1.00000e+00 2.00000e+00
1.00000e+00 2.00000e+00 1.00000e+00
P =
0 2 1
PA =
1.00000e+00 1.00000e+00 0.00000e+00
1.00000e+00 2.00000e+00 1.00000e+00
1.00000e+00 1.00000e+00 2.00000e+00
L =
1.00000e+00 0.00000e+00 0.00000e+00
1.00000e+00 1.00000e+00 0.00000e+00
1.00000e+00 0.00000e+00 2.00000e+00
U =
1.00000e+00 1.00000e+00 0.00000e+00
0.00000e+00 1.00000e+00 1.00000e+00
0.00000e+00 0.00000e+00 1.00000e+00
==========================================
A =
1.00000e+00 5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01
5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01
3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01
2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01
2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01 1.11111e-01
P =
0 1 2 3 4
PA =
1.00000e+00 5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01
5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01
3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01
2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01
2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01 1.11111e-01
L =
1.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
5.00000e-01 8.33333e-02 0.00000e+00 0.00000e+00 0.00000e+00
3.33333e-01 8.33333e-02 5.55556e-03 0.00000e+00 0.00000e+00
2.50000e-01 7.50000e-02 8.33333e-03 3.57143e-04 0.00000e+00
2.00000e-01 6.66667e-02 9.52381e-03 7.14286e-04 2.26757e-05
U =
1.00000e+00 5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01
0.00000e+00 1.00000e+00 1.00000e+00 9.00000e-01 8.00000e-01
0.00000e+00 0.00000e+00 1.00000e+00 1.50000e+00 1.71429e+00
0.00000e+00 0.00000e+00 0.00000e+00 1.00000e+00 2.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 1.00000e+00
```

2. Discussion

Follow the method of solving L and U in class, we know that for every turn, we always solve the diagonal of L first, then solve the off-diagonal term of L on that same column and the off-diagonal term of U on that same row. And the element in L and U is directly related to the element in the same place in A. So if we encounter $l_{ii} = 0$ and swap the row in A, we have to recalculate L and U. We swap the row until all the diagonal are non-zero or zero is at the bottom.

So we have : $LU = PA$

Check the result using MATLAB, we can regenerate A as expected.

- **Inverse of a matrix, using LU-decomposition**

1. Result

Run the code *inverse_matrix.c*.

```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ gcc -o c.out inverse_matrix.c

cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ ./c.out
A =
1.00000e+00 1.00000e+00 0.00000e+00
1.00000e+00 1.00000e+00 2.00000e+00
1.00000e+00 2.00000e+00 1.00000e+00
PAX = LUX = Identity,
X = Inverse of (PA)
1.50000e+00 -1.00000e+00 5.00000e-01
-5.00000e-01 1.00000e+00 -5.00000e-01
-5.00000e-01 0.00000e+00 5.00000e-01
Inverse A =
1.50000e+00 5.00000e-01 -1.00000e+00
-5.00000e-01 -5.00000e-01 1.00000e+00
-5.00000e-01 5.00000e-01 0.00000e+00
CHECK if A * (inverse A) == I
A * (inverse A) - I =
0.00000e+00 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00
0.00000e+00 0.00000e+00 0.00000e+00
===========================================
A =
1.00000e+00 5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01
5.00000e-01 3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01
3.33333e-01 2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01
2.50000e-01 2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01
2.00000e-01 1.66667e-01 1.42857e-01 1.25000e-01 1.11111e-01
PAX = LUX = Identity,
X = Inverse of (PA)
2.50000e+01 -3.00000e+02 1.05000e+03 -1.40000e+03 6.30000e+02
-3.00000e+02 4.80000e+03 -1.89000e+04 2.68800e+04 -1.26000e+04
1.05000e+03 -1.89000e+04 7.93800e+04 -1.17600e+05 5.67000e+04
-1.40000e+03 2.68800e+04 -1.17600e+05 1.79200e+05 -8.82000e+04
6.30000e+02 -1.26000e+04 5.67000e+04 -8.82000e+04 4.41000e+04
Inverse A =
2.50000e+01 -3.00000e+02 1.05000e+03 -1.40000e+03 6.30000e+02
-3.00000e+02 4.80000e+03 -1.89000e+04 2.68800e+04 -1.26000e+04
1.05000e+03 -1.89000e+04 7.93800e+04 -1.17600e+05 5.67000e+04
-1.40000e+03 2.68800e+04 -1.17600e+05 1.79200e+05 -8.82000e+04
6.30000e+02 -1.26000e+04 5.67000e+04 -8.82000e+04 4.41000e+04
CHECK if A * (inverse A) == I
A * (inverse A) - I =
0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00 0.00000e+00
4.26326e-14 4.54747e-13 -1.81899e-12 0.00000e+00 1.81899e-12
1.42109e-14 6.82121e-13 0.00000e+00 3.63798e-12 -1.81899e-12
4.26326e-14 -4.54747e-13 4.54747e-12 -1.81899e-12 9.09495e-13
0.00000e+00 0.00000e+00 -9.09495e-13 -1.81899e-12 0.00000e+00
```

2. Discussion

Originally, we solve $Ax = b$, but if we encounter zero pivots, we permute the rows. So we are actually solving x in equation $PAx = b$ through LU-decomposition. And we can expand the method to

$$PA(x_1 \quad x_2 \quad \cdots) = LU(x_1 \quad x_2 \quad \cdots) = (b_1 \quad b_2 \quad \cdots) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \ddots \end{pmatrix}.$$

By solving each column vector of $PAx_n = b_n$ individually, we can get the inverse of matrix *PA*. But our goal is the inverse of A, thus

$$X = (PA)^{-1} \Rightarrow X = A^{-1}P^{-1}$$

$$\therefore A^{-1} = XP$$

Which means we have to permute the column of X to reach A^{-1}.

In the result, I check the inverse by $AA^{-1} - I$ and see if it is zero matrix. The test matrix (3×3) are zeros, as expected. But the target matrix (5×5) is slightly different from zero ($something \times 10^{-14\sim-12}$), because of the round-off error.

- **Conjugate Gradient Algorithm**

To show that the CG algorithm for $(D^\dagger D)x = b$ in the lecture note can work, we can show how the original algorithm can transform to this one.
Let us first assume we have already solved the equation $Dx = b$ through CG algorithm (original), and obtain a series of $\{x_k\}$, $\{r_k\}$, $\{s_k\}$, $\{p_k\}$, $\{\lambda_k\}$, and $\{\mu_k\}$, which means the iteration tells us the relationship between indices.

(1) Initialize x_0

$r_0 = b - Dx_0$

$p_0 = s_0 = D^\dagger r_0$

(2) Iteration

$$x_{k+1} = x_k + \lambda_k p_k \ , \qquad \lambda_k = \frac{(p_k,\ s_k)}{\|Dp_k\|^2}$$

$$r_{k+1} = b - Dx_{k+1} = r_k - \lambda_k D p_k$$

$$s_{k+1} = D^\dagger r_{k+1} = s_k - \lambda_k D^\dagger D p_k$$

$$p_{k+1} = s_{k+1} + \mu_k p_k \ , \qquad \mu_k = -\frac{(D^\dagger D p_k,\ s_{k+1})}{\|Dp_k\|^2}$$

The two theorem also tell us that:

$$(p_i,\ s_k) = 0 \ \text{ for } \ i < k$$

$$(s_i,\ s_j) = 0 \ \text{ for } \ i \neq j$$

Bring $p_k = s_k + \mu_{k-1} p_{k-1}$ into λ_k, we have

$$\lambda_k = \frac{(s_k + \mu_{k-1} p_{k-1},\ s_k)}{\|Dp_k\|^2} = \frac{(s_k,\ s_k)}{\|Dp_k\|^2}$$

From $r_{k+1} = r_k - \lambda_k D p_k$, we can get

$$D^\dagger D p_k = \frac{D^\dagger r_k - D^\dagger r_{k+1}}{\lambda_k}$$

The index on the RHS is bigger than the LHS, since we view it as a relationship, so it is valid here.
Due to the orthogonality of s_i and the relationship $s_k = D^\dagger r_k$,

$$\mu_k = -\frac{(D^\dagger D p_k,\ s_{k+1})}{\|Dp_k\|^2} = \frac{(s_{k+1} - s_k,\ s_{k+1})}{\lambda_k \cdot \|Dp_k\|^2} = \frac{(s_{k+1},\ s_{k+1})}{\lambda_k \cdot \|Dp_k\|^2}$$

Finally, take from the result of λ_k and μ_k, we get

$$\lambda_k = \frac{(s_k,\ s_k)}{\|Dp_k\|^2} \ \ and \ \ \mu_k = \frac{(s_{k+1},\ s_{k+1})}{(s_k,\ s_k)}$$

Now, if we write the symbol like this,

$$D^\dagger D \to A$$

$$\lambda_k \to \alpha_k$$

$$\mu_k \to \beta_{k+1}$$

$$D^\dagger b \to b'$$

We would find that we no longer need $\{r_k\}$, only $\{s_k\}$ is left, so we write

$$s_k \to r'_k$$

Then the algorithm will be like this,

(1) Initialize x_0

$$s_0 = r'_0 = D^\dagger r_0 = D^\dagger b - D^\dagger D x_0 = b' - A x_0$$

$$p_0 = s_0 = r'_0$$

(2) Iteration

$$x_{k+1} = x_k + \alpha_k p_k \ , \qquad \alpha_k = \frac{(r'_k,\ r'_k)}{\|Dp_k\|^2}$$

$$s_{k+1} = r'_{k+1} = D^\dagger r_{k+1} = r'_k - \alpha_k A p_k$$

$$p_{k+1} = r'_{k+1} + \beta_{k+1} p_k \ , \qquad \beta_k = \frac{(r'_{k+1},\ r'_{k+1})}{(r'_k,\ r'_k)}$$

We get the new form from the original algorithm, and from the new correction vector r'_0, we know that we are solving $D^\dagger D x = b$. Also, from their index relation, we can know that this algorithm is valid, we can get a series of $\{x_k\}$ and finally reach to the answer.

These two algorithm are basically the same, since

$$Dx = b \ \xrightarrow{D^\dagger} \ D^\dagger D x = D^\dagger b$$

$$r_{k+1} = b - D x_{k+1} \ \xrightarrow{D^\dagger} \ D^\dagger r_{k+1} = D^\dagger b - D^\dagger D x_{k+1} = r'_{k+1}$$

We can view the new r'_k as the old s_k, which has the implicit meaning of gradient.

- ## Solve Linear System with Conjugate Gradient

1. Result

Run the code *CG_algorithm.c*, and read input file from *D_Re.txt*, *D_Im.txt*, *b_Re.txt*, *b_Im.txt* we can get the result.

```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ gcc -o d.out CG_algorithm.c
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_6
$ ./d.out
D =
1.00000e+00+0.00000e+00*I 1.00000e+00+9.90000e-01*I 0.00000e+00+0.00000e+00*I 2.49000e-01+0.00000e+00*I 0.00000e+00+0.00000e+00*I
5.00000e-01+0.00000e+00*I 3.30000e-01+0.00000e+00*I 0.00000e+00+1.00000e+00*I 0.00000e+00+0.00000e+00*I 1.23000e-01+0.00000e+00*I
0.00000e+00+0.00000e+00*I 2.12300e+00+1.00000e+00*I 2.15000e-01+0.00000e+00*I 1.31000e-02+0.00000e+00*I 1.27000e-02+0.00000e+00*I
2.64500e-01+0.00000e+00*I 0.00000e+00+0.00000e+00*I 0.00000e+00+0.00000e+00*I 1.30000e-02+0.00000e+00*I 0.00000e+00+0.00000e+00*I
0.00000e+00+0.00000e+00*I 1.00000e-03+0.00000e+00*I 0.00000e+00+0.00000e+00*I 1.00000e+00+-8.97000e+00*I 1.10000e-02+0.00000e+00*I
b =
1.00100e+00+0.00000e+00*I
1.00000e+00+8.77000e-01*I
0.00000e+00+0.00000e+00*I
0.00000e+00+0.00000e+00*I
0.00000e+00+0.00000e+00*I

Solve Dx = b
Iteration : 1 ,  Error = 1.66072e+00
Iteration : 2 ,  Error = 1.42606e+00
Iteration : 3 ,  Error = 1.85543e-01
Iteration : 4 ,  Error = 8.09014e-02
Iteration : 5 ,  Error = 7.40281e-14
x =
x1 = -3.90575e-03 + (2.06941e-04) * I
x2 = 4.97930e-01 + (-4.92109e-01) * I
x3 = -6.98390e+00 + (-1.30948e+00) * I
x4 = 7.94669e-02 + (-4.21046e-03) * I
x5 = -3.83610e+00 + (6.52292e+01) * I
```

2. Discussion

Since the matrix is not Hermitian, I choose to use the first CG Algorithm. And the matrix is only 5x5, so I did not use CG Algorithm with restart.

It needs 5 iterations, which is reasonable and lucky. Since there are 5 degree of freedom in solution *x*; and that the initial $x_0 = 0$ we picked is closed to the solution, so no need of additional adjustment.

If we choose the initial $x_0 = 100$, we will get the result below.

Which give us the same answer, but with one more iteration and with bigger error.

```
Solve Dx = b
Iteration : 1 ,  Error = 1.10217e+05
Iteration : 2 ,  Error = 9.53897e+03
Iteration : 3 ,  Error = 1.34569e+03
Iteration : 4 ,  Error = 3.09090e-01
Iteration : 5 ,  Error = 1.88556e-05
Iteration : 6 ,  Error = 1.04556e-12
x =
x1 = -3.90580e-03 + (2.06882e-04) * I
x2 = 4.97930e-01 + (-4.92109e-01) * I
x3 = -6.98390e+00 + (-1.30948e+00) * I
x4 = 7.94669e-02 + (-4.21045e-03) * I
x5 = -3.83610e+00 + (6.52292e+01) * I
```

Test the CG Algorithm with the matrix in the first question, and find the solution of *x* that brings it to the first column of the identity matrix. It should be the first column of the inverse matrix, as expected.

```
x =
x1 = 2.50000e+01 + (0.00000e+00) * I
x2 = -3.00000e+02 + (0.00000e+00) * I
x3 = 1.05000e+03 + (0.00000e+00) * I
x4 = -1.40000e+03 + (0.00000e+00) * I
x5 = 6.30000e+02 + (0.00000e+00) * I
```