Numerical Analysis For Materials

Homework #3

Gilwoon Lee

ID: 20120083
Dept: Material Science and Engineering

Homework #3

Determining answer and A^{-1}

1. Idea to determine answer and A^{-1}

```
//Inverse matrix 출력을 위한 함수 sequence

Normalization(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Pivoting(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Lower(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Upper(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Normalization(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);

//Inverse matrix 출력을 위한 함수 sequence
Multiple(Matrix_given, Original, Inverse_matrix, row_max, col_max);
```

Gauss-Jordan

Scaled Partial Pivoting 이용

-> Normalization, Pivoting,
Lower, Upper 제거 후 Normalization 순으로 진행
(예외처리는 일부러 진행하지 않고 결과에서 논의함)

1. Code - body

* Description of Program

1. Info. of Program

```
/* ================================================================================
Program: Determining inverse matrix of given matrix
Date: 2015.03.23
Made by Gilwoon Lee
POSTECH, project 3 of [AMSE417] Numerical analysis for materials

Development environment: Visual Studio 2013
Code language: C
================================================================================*/
#define _CRT_SECURE_NO_WARNINGS //Secure warning을 skip하는 명령문

#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#include"Gilwoon.h"
```

Gilwoon.h | AMSE417_HW3_main.c* | AMSE417_HW3_Defalut.c | AMSE417_HW3_Get_data.c | AMSE417_HW3_Gauss-Jordan.c

2. Characteristics of Program

- 반복 여부 선택 가능
- 시간 출력 가능
- 헤더 및 각 기능 함수화
- 크기제한 없음(malloc 이용)

3. Process of main()

Program_explanation() ⇒ Get_data() ⇒ time 출력 ⇒ 재실행여부

```
int main()
{
int choice = 1;//choice: program 반복 여부상수, 1일 경우 실행
clock_t time_start = 0;//시간출력 관련 변수들
clock_t time_end = 0;
double exe_time = 0;

Program_explanation();//Program explanation

while (choice == 1)//program 반복여부 관련 while문
{
Program_info();//Program 한 줄 설명

time_start = clock();

/*정보 scan 작업 함수.
Get_data() 내부에서 Inverse matrix를 구하는 함수 실행*/
Get_data();

//걸린 시간 출력
time_end = clock();
exe_time = (double)(time_end - time_start) / CLOCKS_PER_SEC;
printf("\nexecusion time: %.4f sec\n", exe_time);
printf("============================\n");

//재실행 여부
printf("Enter ""1"" to do again, Enter any number to exit: ");
scanf("%d", &choice);
}

return 0;
}
```

1. Code - body

4. Header

```
#ifndef __GILWOON_H__//헤더가 정의 되지 않았을 때만 아래를 실행
#define __GILWOON_H__//헤더 정의

//defalut 함수 모음
void Program_explanation();
void Program_info();

//Gauss elemination
void Get_data();
void Data_printing(double **Matrix_given, double *Vector_given, double **Inverse_matrix, int row_max, int col_max);
void Normalization(double **Matrix_given, double *Vector_given, double **Inverse_matrix, int row_max, int col_max);
void Pivoting(double **Matrix_given, double *Vector_given, double **Inverse_matrix, int row_max, int col_max);
void Lower(double **Matrix_given, double *Vector_given, double **Inverse_matrix, int row_max, int col_max);
void Upper(double **Matrix_given, double *Vector_given, double **Inverse_matrix, int row_max, int col_max);
void Multiple(double **Matrix_given, double **Original, double **Inverse_matrix, int row_max, int col_max);

#endif
```

앞으로 사용할 함수를 모아놓았다.

1. Code - body 5. Get_data()

```
#define _CRT_SECURE_NO_WARNINGS //Secure warning을 skip하는 명령문

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

//함수 설명에서 Matrix는 방정식의 계수, Vector는 등식의 우변에 해당하
는 matrix를 의미한다.
void Get_data()
{int row = 0;
int col = 0;
int i = 0;
int j = 0;
int row_max = 0;
int col_max = 0;

//Matrix, Vector 정보가 있는 text 파일 열기
FILE *Matrix_info = fopen("Matrix_info.txt", "r");
if (Matrix_info == NULL) // 없다면 경고 문구 출력민 강제 종료
{printf("No file, please try again.");
return 0;}

//Matrix, Vector 크기 정보 scan
fscanf(Matrix_info, "%d %d\n", &row_max, &col_max);

//Matrix의 2-D 공간, Vector의 1-D 공간 생성
i = 0;
double **Matrix_given = (double**)malloc(sizeof(int*)*row_max);
while (i<row_max)
{Matrix_given[i++] = (double*)malloc(sizeof(int)*col_max);}
i = 0;
double **Original = (double**)malloc(sizeof(int*)*row_max);
while (i<row_max)
{Original[i++] = (double*)malloc(sizeof(int)*col_max);}
i = 0;
double **Inverse_matrix = (double**)malloc(sizeof(int*)*row_max);
while (i<row_max)
{Inverse_matrix[i++] = (double*)malloc(sizeof(int)*col_max);}
double *Vector_given = (double*)malloc(sizeof(int*)*row_max);
```

```
//Matrix 정보 scan
while (row<row_max)
{while (col<col_max)
{fscanf(Matrix_info, "%lf", &Matrix_given[row][col++]);}
col = 0;
row++;}

row = 0;
col = 0;
//Matrix original 정보 저장
while (row<row_max)
{while (col<col_max)
{Original[row][col] = Matrix_given[row][col++];}
col = 0;
row++;}

// Vector 정보 scan
row = 0;
col = 0;
while (row<row_max)
{fscanf(Matrix_info, "%lf", &Vector_given[row++]);}

//Matrix, Vector 정보가 있는 text 파일 닫기
fclose(Matrix_info);

i = 0;
j = 0;
//Inverse_matrix
while (i < row_max)
{while (j < col_max)
{if (i == j)
{Inverse_matrix[i][j++] = (double)1.0;}
else
{Inverse_matrix[i][j++] = (double)0.0;}}
j = 0;
i++;}
```

1. Code - body

5. Get_data() (이어서)

```
//정보 체크를 위한 출력
printf("<Raw data>\n");
Data_printing(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);

//Inverse matrix 출력을 위한 함수 sequence

Normalization(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Pivoting(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Lower(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Upper(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
Normalization(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);

//Inverse matrix 출력을 위한 함수 sequence
Multiple(Matrix_given, Original, Inverse_matrix, row_max, col_max);

//Matrix의 2-D 공간, Vector의 1-D 공간 free
free(Matrix_given);
}
```

원하는 함수만 골라서 실행 가능하다

6. Data_printing

```
void Data_printing(double **Matrix_given, double *Vector_given, double
**Inverse_matrix, int row_max, int col_max)
{
int i = 0;
int j = 0;
while (i < row_max)
{
while (j < col_max)
{
printf("%f ", Matrix_given[i][j++]);
}
printf("%f\n", Vector_given[i]);
j = 0;
i++;
}
printf("\n");

i = 0;
j = 0;
while (i < row_max)
{
while (j < col_max)
{
printf("%f ", Inverse_matrix[i][j++]);
}
printf("\n");
j = 0;
i++;
}
printf("\n");
}
```

1. Code - body

7. Normalization()

```
void Normalization(double **Matrix_given, double *Vector_given, double
**Inverse_matrix, int row_max, int col_max)
{
double max_in_row = 0;
int pivot_row = 0;
int pivot_col = 0;
while (pivot_row < row_max)
{
max_in_row = 0;
while (pivot_col < col_max)
{
if (fabs(max_in_row) < fabs(Matrix_given[pivot_row][pivot_col]))
{
max_in_row = Matrix_given[pivot_row][pivot_col];
}
pivot_col++;
}
pivot_col = 0;
while (pivot_col < col_max)
{
Matrix_given[pivot_row][pivot_col] /= max_in_row;
Inverse_matrix[pivot_row][pivot_col++] /= max_in_row;
}
Vector_given[pivot_row] /= max_in_row;
pivot_col = 0;
pivot_row++;
}

printf("<Normalization>\n");
Data_printing(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
}
```

8. Pivoting()

```
void Pivoting(double **Matrix_given, double *Vector_given, double
**Inverse_matrix, int row_max, int col_max)
{double max_in_col = 0;
double temp;
int pivot_row = 0;
int pivot_col = 0;
int pivot_row_max = 0;
int pivot_row_temp = 0;
while (pivot_col < col_max - 1)
{max_in_col = 0;
pivot_row_temp = pivot_row;
while (pivot_row_temp < row_max)
{if (max_in_col < fabs(Matrix_given[pivot_row_temp][pivot_col]))
{max_in_col = fabs(Matrix_given[pivot_row_temp][pivot_col]);
pivot_row_max = pivot_row_temp;}
pivot_row_temp++;}
pivot_col = 0;
while (pivot_col < col_max)
{temp = Matrix_given[pivot_row][pivot_col];
Matrix_given[pivot_row][pivot_col] = Matrix_given[pivot_row_max][pivot_col];
Matrix_given[pivot_row_max][pivot_col] = temp;
temp = Inverse_matrix[pivot_row][pivot_col];
Inverse_matrix[pivot_row][pivot_col] = Inverse_matrix[pivot_row_max][pivot_col];
Inverse_matrix[pivot_row_max][pivot_col++] = temp;}
temp = Vector_given[pivot_row];
Vector_given[pivot_row] = Vector_given[pivot_row_max];
Vector_given[pivot_row_max] = temp;
pivot_row++;
pivot_col++;}

printf("<Pivoting>\n");
Data_printing(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);}
```

1. Code - body

9. Lower(), Upper()

```
void Lower(double **Matrix_given, double *Vector_given, double **Inverse_matrix,
int row_max, int col_max)
{
double ratio;
int pivot_row = 0;
int pivot_col = 0;
int pivot_row_temp = 0;
pivot_row = 0;
pivot_col = 0;
while (pivot_row < row_max - 1)
{
pivot_row_temp = pivot_row + 1;

while (pivot_row_temp < row_max)
{
ratio = Matrix_given[pivot_row_temp][pivot_row] /
Matrix_given[pivot_row][pivot_row];
while (pivot_col < col_max)
{
Matrix_given[pivot_row_temp][pivot_col] -= Matrix_given[pivot_row][pivot_col] *
ratio;
Inverse_matrix[pivot_row_temp][pivot_col] -=
Inverse_matrix[pivot_row][pivot_col] * ratio;
pivot_col++;
}
Vector_given[pivot_row_temp] -= Vector_given[pivot_row] * ratio;
pivot_col = 0;
pivot_row_temp++;
}
pivot_row++;
}

printf("<Lower matrix elimination>\n");
Data_printing(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
}
```

```
void Upper(double **Matrix_given, double *Vector_given, double **Inverse_matrix,
int row_max, int col_max)
{
double ratio;
int pivot_row = 0;
int pivot_col = 0;
int pivot_row_temp = 0;
while (row_max - 1 - pivot_row > 0)
{
pivot_row_temp = pivot_row + 1;

while (row_max - 1 - pivot_row_temp > row_max - 1 - row_max)
{
ratio = Matrix_given[row_max - 1 - pivot_row_temp][row_max - 1 - pivot_row] /
Matrix_given[row_max - 1 - pivot_row][row_max - 1 - pivot_row];
while (col_max - 1 - pivot_col > -1)
{
Matrix_given[row_max - 1 - pivot_row_temp][col_max - 1 - pivot_col] -=
Matrix_given[row_max - 1 - pivot_row][col_max - 1 - pivot_col] * ratio;
Inverse_matrix[row_max - 1 - pivot_row_temp][col_max - 1 - pivot_col] -=
Inverse_matrix[row_max - 1 - pivot_row][col_max - 1 - pivot_col] * ratio;
pivot_col++;
}
Vector_given[row_max - 1 - pivot_row_temp] -= Vector_given[row_max - 1 -
pivot_row] * ratio;
pivot_col = 0;
pivot_row_temp++;
}
pivot_row++;
}

printf("<Upper matrix elimination>\n");
Data_printing(Matrix_given, Vector_given, Inverse_matrix, row_max, col_max);
}
```

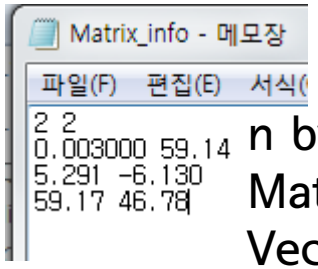
1. Code - body

10. Multiple()

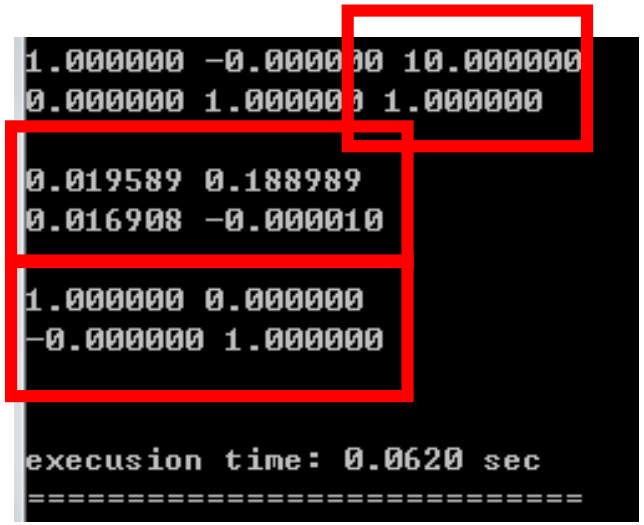
```
void Multiple(double **Matrix_given, double **Original, double **Inverse_matrix,
int row_max, int col_max)
{
int i = 0;
int j = 0;
int k = 0;

while (i < row_max)
{
while (j < row_max)
{
Matrix_given[i][j++] = 0;
}
j = 0;
i++;
}

i = 0;
j = 0;
while (i < row_max)
{
while (k < row_max)
{
while (j < row_max)
{
Matrix_given[i][k] += Original[i][j] * Inverse_matrix[j++][k];
}
j = 0;
k++;
}
k = 0;
i++;
}

i = 0;
j = 0;
while (i < row_max)
{
while (j < col_max)
{
printf("%f ", Matrix_given[i][j++]);
}
j = 0;
i++;
printf("\n");
}
printf("\n");
}
```

2. Result

1. Round of error가 나는 경우



n by n Matrix
Vector

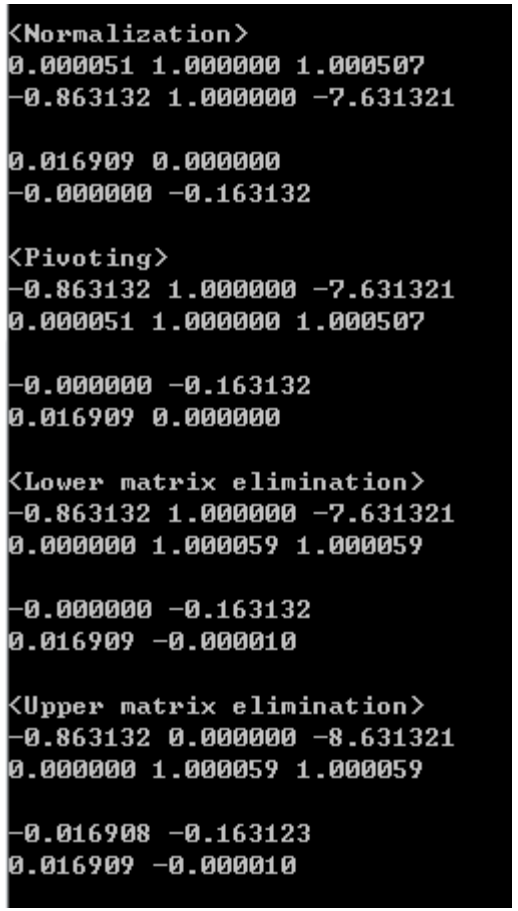


Solution

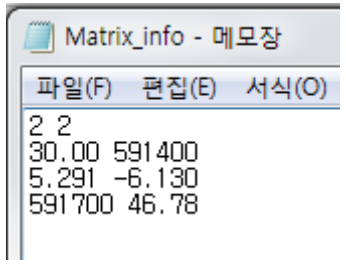
Inverse matrix

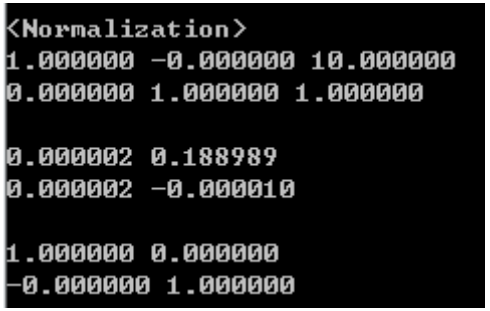
Proof (should be I)

Procedure



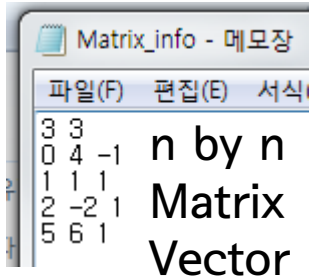
Solution의 경우 Pivoting을 통해 큰 수로 나누어주는 작업을 통해 error를 줄였다.
값이 10과 1로 정확하게 나오는 것을 확인하였다.
아래와 같이 수가 커지는 경우에도 값을 정확히 보여준다.



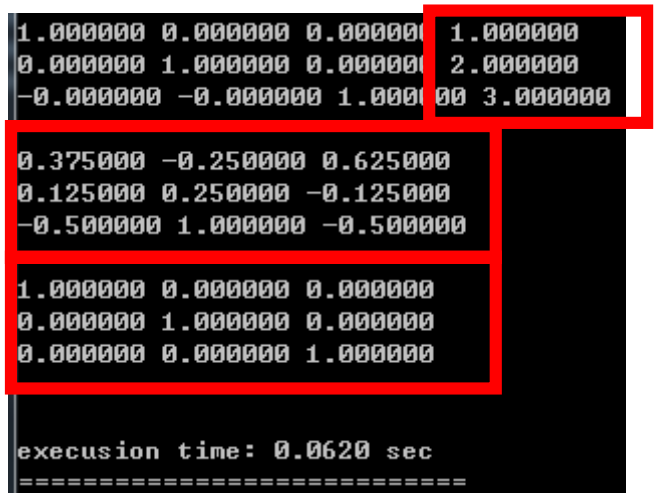


2. Result

2. (k, k)=0인 경우



n by n Matrix
Vector



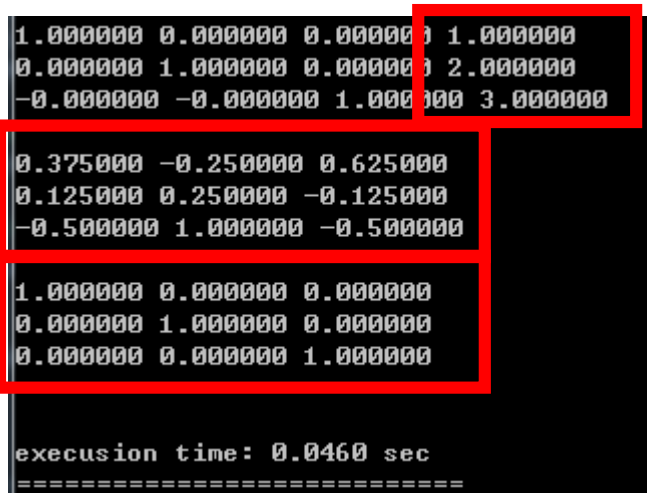
Solution

Inverse matrix

Proof (should be I)

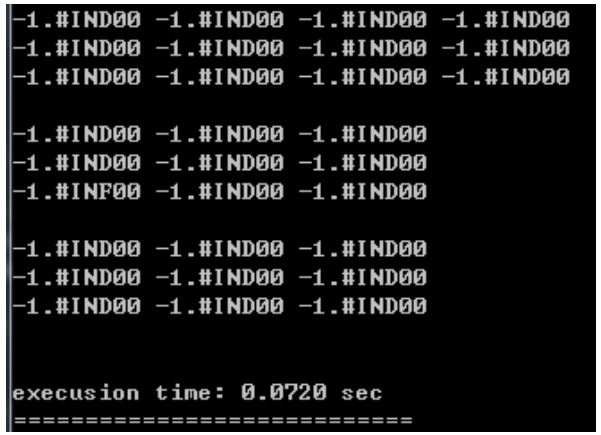
Normalization -> Pivoting

Solution의 경우 Pivoting을 하여 구하더라도 정답이 나오는 것을 확인할 수 있었다.
Inverse matrix의 또한 검증을 해본 결과 pivoting과 관계 없이 역행렬을 구할 수 있음을 알 수 있다.



Pivoting -> Normalization

반대 순서로 하더라도 결과는 같게 나오는 것을 확인하였다. 몇 번 시도하였을 때, 이 경우가 전자에 비해 시간이 적게 나오는 경향성을 보여주었다.

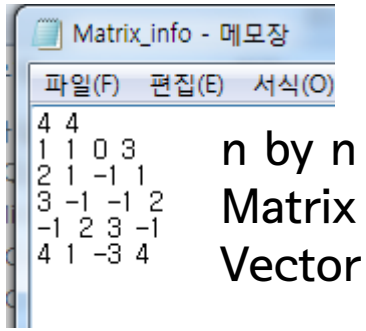


No Pivoting

0으로 나누는 작업을 실행하여 값이 나오지 않음을 확인하였다.
즉, inverse matrix는 아니더라도 정답을 구하기 위해서는 pivoting이 필요함을 알 수 있다.

2. Result

3. 4 by 4 행렬



n by n Matrix Vector

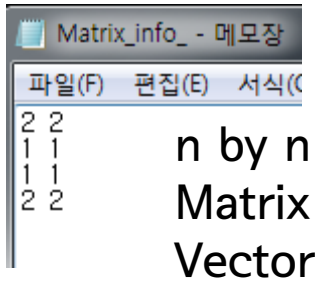
```
1.000000 0.000000 0.000000 0.000000 -1.000000
0.000000 1.000000 0.000000 0.000000 2.000000
0.000000 0.000000 1.000000 0.000000 0.000000
-0.000000 -0.000000 -0.000000 1.000000 1.000000

-0.230769 0.205128 0.333333 0.179487
0.076923 0.487179 -0.333333 0.051282
0.000000 -0.333333 0.333333 0.333333
0.384615 -0.230769 0.000000 -0.076923

1.000000 0.000000 -0.000000 0.000000
0.000000 1.000000 -0.000000 -0.000000
-0.000000 -0.000000 1.000000 0.000000
0.000000 -0.000000 -0.000000 1.000000


execusion time: 0.0930 sec
============================
```

Pivoting -> Normalization

Solution의 경우 크기가 커지더라도 정답이 정확하게 나오는 것을 확인할 수 있었다.
Inverse matrix 또한 검증을 해본 결과 정확하게 계산이 가능함을 알 수 있었다.

2. Result

4. Determinant = 0



n by n Matrix Vector

```
<Lower matrix elimination>
1.000000 1.000000 2.000000
0.000000 0.000000 0.000000

1.000000 0.000000
-1.000000 1.000000

<Upper matrix elimination>
-1.#IND00 -1.#IND00 -1.#IND00
0.000000 0.000000 0.000000

1.#INF00 -1.#INF00
-1.000000 1.000000

<Normalization>
-1.#IND00 -1.#IND00 -1.#IND00
-1.#IND00 -1.#IND00 -1.#IND00

1.#INF00 -1.#INF00
-1.#INF00 1.#INF00

-1.#IND00 -1.#IND00
-1.#IND00 -1.#IND00


execusion time: 0.0620 sec
============================
Enter 1 to do again, Enter any number to exit:
```

Pivoting -> Normalization

Diagonal element가 어떻게 해도 0이 되는 경우로 나타났다.
즉, 이렇게 pivoting에도 불구하고 lower, upper matrix의 계산 결과에서 0으로 나누는 에러가 나는 것을 확인할 경우 determinant가 0임을 확인할 수 있으며, 이 경우 직접 해가 없는지, 무수히 많은지를 직접 계산해야만 한다.
(실제 계산 결과가 어떻게 되는지 확인하기 위하여 일부러 예외처리를 하지 않았습니다.)

3. Conclusion

* Problem answer

Determinant가 0이 되는 경우를 제외하고는 Scaled Partial Pivoting을 이용하여 모든 inverse matrix를 성공적으로 구할 수 있었으며, 선형 연립방정식의 정답도 잘 구할 수 있었다.

* 방법의 속도(횟수, 시간) 비교

1) n by n matrix의 크기가 커지면 커질 수록 계산 횟수가 많아져 시간이 증가하는 경향성이 있다.
2) 1초보다 훨씬 작은 시간 내에 값을 구하는 것을 확인할 수 있었다.
3) Determinant가 0이 되는10 by 10을 계산하여 결과는 확인 불가능 하였지만, 대략적인 계산 소요시대는 확인할 수 있었다. 연산 수에 의존하는 함수라고 생각되나 정확한 비례라기 보다는 수가 커질 수록 기하급수적으로 늘어날 것으로 예상된다. (2, 3, 4인 경우와 견주어보았을 때)

Gauss-Jordan, Scaled Partial Pivoting, 성공적.


Thank
you!