Homework#3

2019030991 홍정범

1. Solve the following set of equations

1. Gauss-Jordan Elimination

```
printf("\n1. Gauss Jordan Elimination\n\n");
gaussj(copyA,n,copyb,n);

// 해 출력
for(int i = 1; i <= n; i++) {
  printf("%f ", copyb[i][1]);
}
printf("\n");

// 복사본 만들기
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        copyA[i][j] = A[i][j];
    }
}

for (int i = 1; i <= n; i++) {
    copyb[i][1] = b[i][1];
}
```

가우스 조던의 경우 그냥 함수만 실행시키고 답만 확인하면 되어서 간단하게 구성했다.

2. LU Decomposition

```c
printf("\n2. LU Decompostion\n\n");

float det = 0.f;
int* indx = allocateIntVector(10);
float tempb[10];

for (int i = 1; i <= n; i++) {
        tempb[i] = b[i][1];
}

ludcmp(copyA,n,indx,&det);
lubksb(copyA,n,indx,tempb);

// 해 출력
for(int i = 1; i <= n; i++) {
    printf("%.20f ", tempb[i]);
    mprove_x[i] = tempb[i];
}
printf("\n");
```

LU decomposition의 경우에도 수업시간에 수도 코드에서 보았던 것처럼 구성하면 된다. 거의 할 일이 함수 호출밖에 없기 때문에 간단하다.

3. SVD

```
printf("\n3. Singular Value Decompostion\n\n");

float w[10];
float** v = allocateMatrix(10,10);

svdcmp(copyA,n,n,w,v);

float** ut = allocateMatrix(10,10);
transposeMatrix(copyA,ut,n,n);

float** temp = allocateMatrix(10,10);
float** coef = allocateMatrix(10,10);

for(int i = 1; i <= n; i++) {
  if(w[i] < 1e-6) w[i] = 0.0;
  else w[i] = 1 / w[i];
}

float** ww = allocateMatrix(10,10);
for(int i = 1; i <= n; i++) {
  for(int j = 1; j <= n; j++) {
    if(i == j) ww[i][j] = w[i];
    else ww[i][j] = 0;
  }
}
```

SVD가 제법 어려웠는데, 우선 svdcmp를 호출해서 U,V,W^T를 각각 완성해준다. 원래 식이 Ax = b 즉 UWV^T * x = b 이기 때문에 x를 구하기 위해선 각각 역행렬을 구해줘야 한다. x = V[1/W]U^T * b 에 식을 이용해서 하면된다. 수업시간에 배운 내용이기도 하고 피피티에 나와있는 내용이라서 이를 구현해내기만 하면 되었다. 우선 U^T가 필요하기 때문에 연산했다. 그리고 w도 대각선 성분이 0으로 나누어진 행렬의 꼴로 변환해주었다. 물론 SVD로 근사할 때 singular 해서 원래 대각선에 0이 들어가 있을 때는 그냥 0으로 해줬다.

```c
matrixMultiply(v,ww,temp,n,n,n);

matrixMultiply(temp,ut,coef,n,n,n);

float** answer = allocateMatrix(10,10);

matrixMultiply(coef,b,answer,n,n,1);

for(int i = 1; i <= n; i++) {
  printf("%f ", answer[i][1]);
}
printf("\n");

// 복사본 만들기
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        copyA[i][j] = A[i][j];
    }
}

for (int i = 1; i <= n; i++) {
    copyb[i][1] = b[i][1];
}
```

그 이후로는 내가 자체적으로 만들어둔 행렬 곱하기 연산을 이용해서 순서대로 곱해주었다. 그리고 출력하면 완료. 아래 사진은 lineq2.dat 의 출력 값을 나타낸다.

```
❯ ./runMain
filename? : lineq2.dat

1. Gauss Jordan Elimination

-2.873565 -0.612356 0.976277 0.635818 -0.553441

2. LU Decompostion

-2.87356615066528320312 -0.61235666275024414062 0.97627735137939453125 0.63581860065460205078 -0.55344110727310180664

after mprove..
-2.87356615066528320312 -0.61235660314559936523 0.97627735137939453125 0.63581854104995727539 -0.55344104766845703125

3. Singular Value Decompostion

-2.873564 -0.612357 0.976277 0.635818 -0.553440
```

lineq1.dat의 경우 singular matrix여서 1번과 2번의 경우 연산이 되지 않는다. 아래는 내 코드중 일부를 주석처리하고 실행했을 때를 보여 준다.

```
❯ ./runMain
filename? : lineq1.dat

1. Gauss Jordan Elimination

Numerical Recipes run-time error...
gaussj: Singular Matrix
...now exiting to system...
```

```
❯ ./runMain
filename? : lineq1.dat

2. LU Decompostion

Numerical Recipes run-time error...
Singular matrix in routine ludcmp
...now exiting to system...
```

```
../NRs/ansi/other/nr.h:184:7: note: 'fr
1 warning generated.
❯ ./runMain
filename? : lineq1.dat

3. Singular Value Decompostion

1.733333 -1.533333 -0.200000 -0.733333
```

소수 표시, 유효숫자의 수: 15

$$\begin{pmatrix} 4 & 2 & 3 & -1 \\ -2 & -1 & -2 & 2 \\ 5 & 3 & 4 & -1 \\ 11 & 4 & 6 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1.733333 \\ -1.533333 \\ -0.200000 \\ -0.733333 \end{pmatrix} = \begin{pmatrix} 3.999999 \\ -2.999999 \\ 3.999999 \\ 10.999998 \end{pmatrix}$$

2. Apply the method of iterative improvement(mprove()) to the above problem and discuss the results

```c
printf("\nafter mprove..\n");

mprove(A,copyA,n,indx,mprove_b,mprove_x);

for(int i = 1; i <= n; i++) {
  printf("%.20f ", mprove_x[i]);
}
printf("\n");
```

lu decompostion을 하면서 바로 최적화를 진행했다.

결과값이 미세하게 차이나서 출력 범위를 늘렸다.

```
2. LU Decompostion

-2.87356615066528320312 -0.61235666275024414062 0.97627735137939453125 0.63581860065460205078 -0.55344110727310180664

after mprove..
-2.87356615066528320312 -0.61235660314559936523 0.97627735137939453125 0.63581854104995727539 -0.55344104766845703125
```

답:

$$x_1=-2.87356621480709071 95$$

$$x_2=-0.6123566214807090 7195$$

$$x_3=0.9\overline{76277372}26277372263$$

$$x_4=0.635818561001042752 87$$

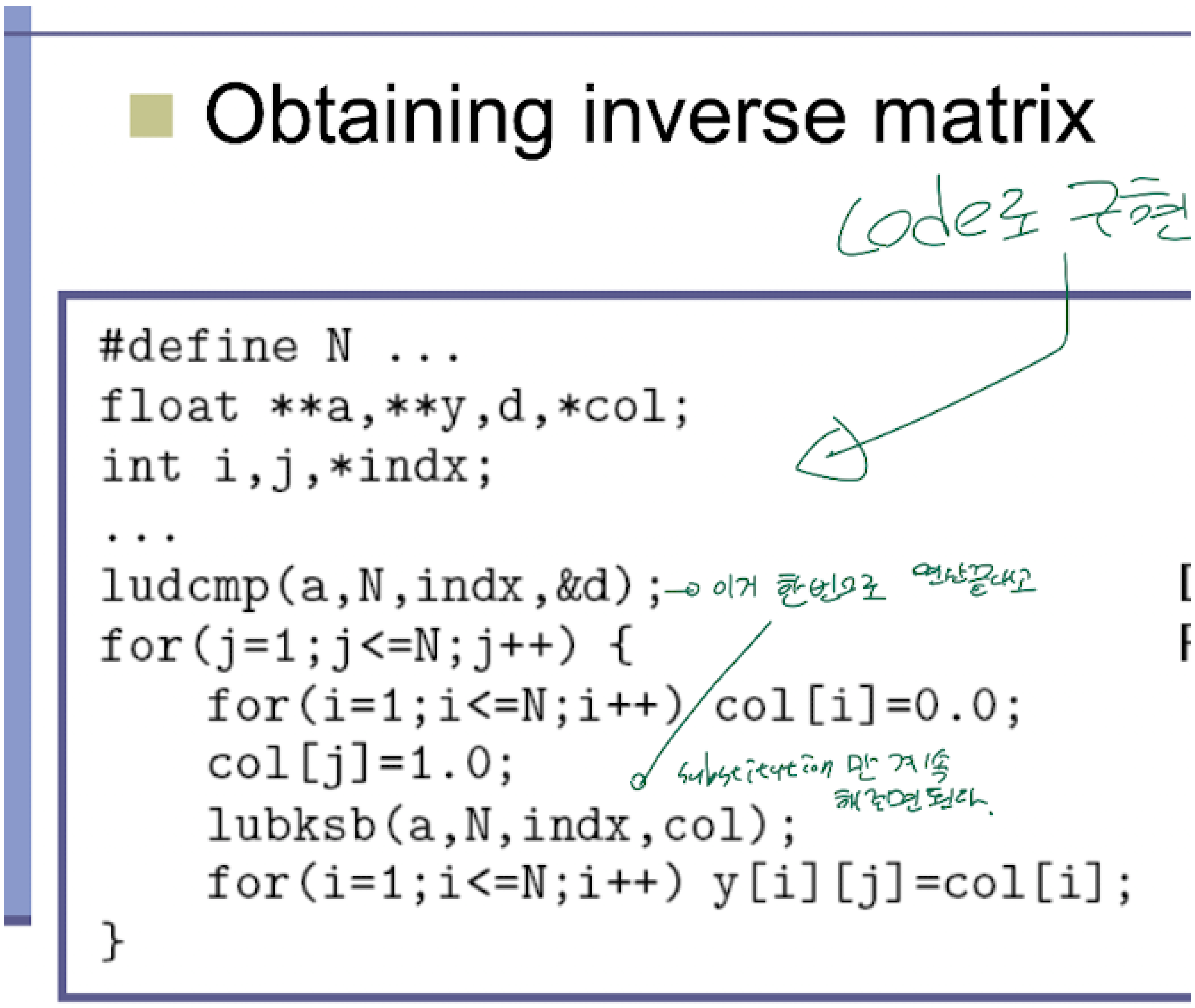
$$x_5=-0.553441084462982273 20$$

$$\text{일반해}:X=\begin{pmatrix}-2.87356621480709071 95\\-0.6123566214807090 7195\\0.9\overline{76277372}26277372263\\0.635818561001042752 87\\-0.553441084462982273 20\end{pmatrix}$$

답에 미세하게 가까워지는 경향을 보인다.

---

3. Find the inverse and the determinant of the matrix Ai in the above problem



해당 내용은 수치해석 본 강의 ppt 에 들어있던 lu decomposition의 결과 값을 이용하여 inverser matrix를 구하는 코드를 참고하여 작성했다.

```c
printf("\n4. Inverse Matrix and Determinant\n\n");

float* col = allocateFloatVector(10);
float** inverse = allocateMatrix(10,10);
int* new_indx = allocateIntVector(10);

ludcmp(A,n,new_indx,&det);
for(int j = 1; j <= n; j++) det *= A[j][j];
printf("determinant : %f\n\n", det);

printf("inverse matrix : \n\n");
for(int j = 1; j <= n; j++) {
    for(int i = 1; i <= n; i++) col[i] = 0.0;
    col[j] = 1.0;
    lubksb(A,n,indx,col);
    for(int i = 1; i <= n; i++) inverse[i][j] = col[i];
}

for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        printf("%f ", inverse[i][j]);
    }
    printf("\n");
}
```

lineq2.dat의 결과 값으로 올바른 출력임을 확인했다.

```
4. Inverse Matrix and Determinant

determinant : 3835.999512

inverse matrix :

0.354536 0.766945 0.207769 -0.595412 0.253128
0.035454 0.126695 0.195777 -0.159541 0.050313
-0.138686 -0.098540 -0.096715 0.124088 0.016423
-0.052138 -0.303962 -0.023201 0.234619 -0.044578
0.149114 0.459333 0.051356 -0.171011 0.042492
```

☑ 소수 표시, 유효숫자의 수: 2

$$\begin{vmatrix} 2 & -4 & -5 & 5 & 0 \\ -1 & 1 & 2 & 0 & 4 \\ -1 & 6 & 0 & 3 & 2 \\ 0 & 1 & 3 & 7 & 5 \\ 5 & 0 & 8 & 7 & -2 \end{vmatrix} = 3836$$

$$\begin{pmatrix} 2 & -4 & -5 & 5 & 0 \\ -1 & 1 & 2 & 0 & 4 \\ -1 & 6 & 0 & 3 & 2 \\ 0 & 1 & 3 & 7 & 5 \\ 5 & 0 & 8 & 7 & -2 \end{pmatrix}^{(-1)} =$$

$$\begin{pmatrix} 0.35453597497393117831 & 0.76694473409801876955 & 0.20776850886339937435 & -0.59541188738269030240 & 0.25312825860271115746 \\ 0.035453597497393117831 & 0.12669447340980187696 & 0.19577685088633993743 & -0.15954118873826903024 & 0.050312825860271115746 \\ -0.\overline{13868613}138686131387 & -0.0\overline{98540145}985401459854 & -0.09\overline{67153284}67153284672 & 0.\overline{12408759}124087591241 & 0.01\overline{64233576}64233576642 \\ -0.052137643378519290928 & -0.30396246089676746611 & -0.023201251303441084463 & 0.23461939520333680918 & -0.044577685088633993743 \\ 0.14911366006256517205 & 0.45933263816475495308 & 0.051355578727841501564 & -0.17101147028154327424 & 0.042492179353493222106 \end{pmatrix}$$

위의 사진은 나의 결과값, 아래 사진은 구글 행렬 계산기를 이용한 결과 값이다.