**NA-HW #05**

2015004693\_양상헌

**Goal:**

* **Gauss-Jordan Elimination [gaussj()]**
* **LU Decomposition [ludcmp(), lubksb()]**
* **Singular Value Decomposiiton [svdcmp(), svbksb()]**
* **Iterative Improvement [mprove()]**

위 4가지 Methods를 이용하여 “lineq1.dat”, “lineq2.dat”, “lineq3.dat” 의 data 파일에 “A”-Matrix와 ”b”-Vector 형태로 입력 되어있는 Linear Equations의 해를 구하고, 각각 Matrix의 Inverse Matrix와 Determinant를 구하고 위의 Methods의 특징과 장단점을 알아본다.

**Background Knowledge:**

**Gauss-Jordan Elimination Method**는 다른 Methods와 비교했을 때 비교적 가장 간단한 방법이라 할 수 있다. “**A**”-Matrix와 “**b**”-Vector를 Gauss Reduction을 계속해서 실행하여 “**A**”-Matrix 를 “**I**”-Identity Matrix 로 만들고 “**b**”-Vector를 “**x**”-Solution Vector로 만드는 방법이다. 기호로 표현하면 **[A : b] -> [I : x]** 의 형태로 표현할 수 있다. 또한 Gauss-Jordan Elimination Method는 Inverse Matrix를 구하는 것에 매우 효과적인데, 위에서 **[A : b]** 형태에서 Gauss Reduction을 하여 **[I : x]** 형태로 만들어 “**x**”-Solution Vector를 구한 것처럼 **[A : I]** 의 형태로 시작하여 Gauss Reduction을 실행하면 위와 같은 방법으로 **[I : A^-1]** 결과를 내고 이를 이용하여 쉽게 Inverse Matrix를 구할 수 있다.

**LU Decomposition Method**는 “**A**”-Matrix를 Lower-triangular Matrix(**L**-Matrix)와 Upper-triangular Matrix(**U**-Matrix)로 분할한 다음 “**b**”-vector 에 대해 Back-Substitution을 각각 **U**-Matrix와 **L**-Matrix에 두 번 하여 “**x**”-Solution Vector를 구하는 Method 이다. **L**-Matrix는 **L[i][j] (i<j)** 값을 모두 0으로 하고, **U**-Matrix는 **U[i][j] (i>j)** 값을 모두 0으로 가진다. **L**-Matrix와 **U**-matirx는 서로 Diagonal Elements (U[i][i], L[i][i])를 각각 모두 1이거나 혹은 weight값을 갖는 상수로 한다. Numerical Recipe에 있는 “ludcmp()”에서는 **L**-Matrix의 diagonal Elements가 모두 1이고, **U**- Matrix의 diagonal Elements 가 각각 weight값을 갖는 숫자로 이뤄져 있다. 이때 **U**-Matrix의 diagonal elements들의 곱은 본래의 **A**-Matrix의 **Determinant**가 된다. **Ax = b** 형태의 linear Equations를 **LUx’ = b’** 형태로 바꾸어 푸는 것이다. 이때의 장점은 Gauss-jordan에 비해 **b**의 값이 계속해서 바뀔 경우 **LU**-decomposition은 한번만 하고 그 뒤론 Back-Substitution 만 해주면 되기 때문에 이러한 경우 더 빠르게 계산을 할 수 있다는 점이다.

**Singular Value Decomposition Method**는 **A**-Matrix를 **U**-Matrix **W**-Matrix **V^T**-Matrix로 분할하여 해를 구하는 방법이다. 여기서 **U**-Matrix와 **V**-Matrix는 orthonormal한 Matrix로 **U**와 **V**의 inverse matrix는 각각의 transpose matrix와 같다. 또한 **W**-Matrix는 Diagonal Matrix로서 Diagonal Elements를 제외한 나머지 Elements는 모두 0인 Matrix이다. 기하학적인 의미로는 **U**-Matrix와 **V**-matrix는 크기 변화에 영향을 미치지 않는 Determinant가 1인 Matrix이고, **W**-Matrix만이 크기 변화에 영향을 미치는 고유한 Determinant값을 갖는 Matrix이다. SVD를 이용하여 Linear Equations의 해를 구할 때 **A = U W V^T** 이므로 **A^-1 = V W^-1 U^T** 로 쉽게 계산 할 수 있다. **U** 와 **V**는 Orthonormal 하기 때문에 쉽게 inverse Matrix를 구할 수 있고, **W**의 경우 diagonal Matrix이기 때문에 각 Diagonal Elements 의 역수를 취해주면 이 역시 쉽게 inverse matrix를 구할 수 있다. 또한 만약 **A**-Matrix가 Singular Matrix인 경우 **W**의 Diagonal Elements중 0 이 존재하게 되는데 이러한 경우 **W**의 inverse Matrix를 구할 때 1/0이 아닌 그냥 0으로 해주어 해를 구할 수 없는 Singular Matrix의 경우에도 근사한 해를 구할 수 있게 해준다.

**Iterative Improvement**는 **A**의 inverse의 크기가 매우 큰 경우 **ill-condition**이 발생하는데 이때 생기는 오차를 계산하여 줄여주는 방법이다. 실제 Equation에서는 **[A]{X} = {B}** 이므로 **{B}-[A]{X} = {0}** 이지만, 계산으로 구한 **{X}**값을 **{X’}**이라 했을 때 **{R} = {B}-[A]{X’}** 이라는 오차가 발생하게 된다. 이를 더 자세히 표현하면 **{R} = [A]{ {X}-{X’} }** 로 표현할 수 있다. 이 식을 **[A^-1]{R} = {X}-{X’}** 으로 나타냈을 때 **{R}**의 크기와 상관없이 **[A^-1]**의 크기가 매우 크다면 **{X}-{X’}**가 크다는 이야기 이고 이를 쉽게 설명하면 **{R}** 이 작다고 해서 항상 **{X’}**를 잘 구한 것은 아니라는 이야기이다. 이 때문에 **{X}-{X’}**값**([A^-1]{R})**이 작아질 때까지 (converge)할 때까지 계속해서 Iteration을 반복하여 오차를 줄이는 방법이다.

**Process:**

**1.**

**Gauss-Jordan Elimination Method**는 기존 “xgaussj.c” 소스코드를 편집하여 실행하였다. 이때 기존과 다르게 *“lineq1.dat”, “lineq2.dat”, “lineq3.dat”*를 한번에 다 읽어오게 하였고, 원래의 **A**-Matrix와 **b**-Vector를 출력하는 부분을 추가하였다. **A**-Matrix의 inverse를 구해 출력하도록 하고, 이로 인해 구해진 **x**-Solution Vector를 출력하도록 해주었다.

*편집된< xgaussj.c >, (편집된 부분 중 중요한 부분은 빨간색으로 표시)*

|  |
| --- |
| /\* Driver for routine gaussj \*/  #include <stdio.h>  #include <stdlib.h>  #define NRANSI  #include "nr.h"  #include "nrutil.h"  #define NP 20  #define MP 20  #define MAXSTR 80  **int** main(**void**)  {  **int** i,j,k,l,m,n;  **float** \*\*a,\*\*ai,\*\*u,\*\*b,\*\*x,\*\*t;  //char dummy[MAXSTR];  FILE \*fp[3];  a=matrix(1,NP,1,NP);  ai=matrix(1,NP,1,NP);  u=matrix(1,NP,1,NP);  b=matrix(1,NP,1,MP);  x=matrix(1,NP,1,MP);  t=matrix(1,NP,1,MP);  **if** ((fp[0] = fopen("lineq1.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[1] = fopen("lineq2.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[2] = fopen("lineq3.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");    **for**(i=0;i<3;i++){  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq%d.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",i+1);  //while (!feof(fp)) {  //fgets(dummy,MAXSTR,fp);  //fgets(dummy,MAXSTR,fp);  fscanf(fp[i],"%d",&n);  fscanf(fp[i],"%d",&n);  //fgets(dummy,MAXSTR,fp);  m = 1;  **for** (k=1;k<=n;k++)  **for** (l=1;l<=n;l++) fscanf(fp[i],"%f ",&a[k][l]);  //fgets(dummy,MAXSTR,fp);  **for** (l=1;l<=m;l++)  **for** (k=1;k<=n;k++) fscanf(fp[i],"%f ",&b[k][l]);  /\* save matrices for later testing of results \*/  **for** (l=1;l<=n;l++) {  **for** (k=1;k<=n;k++) ai[k][l]=a[k][l];  **for** (k=1;k<=m;k++) x[l][k]=b[l][k];  }  //----print original A & B---//  printf("\nOriginal matrix a : \n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",a[k][l]);  printf("\n");  }    printf("\nOriginal matrix b : \n");  **for** (k=1;k<=n;k++) {  printf("%12.6f",b[k][1]);  printf("\n");  }  /\* invert matrix \*/  gaussj(ai,n,x,m);  printf("\nInverse of matrix a : \n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",ai[k][l]);  printf("\n");  }  /\* check inverse \*/  printf("\na times a-inverse:\n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) {  u[k][l]=0.0;  **for** (j=1;j<=n;j++)  u[k][l] += (a[k][j]\*ai[j][l]);  }  **for** (l=1;l<=n;l++) printf("%12.6f",u[k][l]);  printf("\n");  }    //--------edited--------//  printf("\n<Solution vector X> : \n");  **for**(k=1;k<=n;k++){  printf("%12.6f\n",x[k][1]);  }  //----------------------//    /\* check vector solutions \*/  printf("\nCheck the following for equality:\n");  printf("%21s %14s\n","original","matrix\*sol'n");  **for** (l=1;l<=m;l++) {  printf("vector %2d: \n",l);  **for** (k=1;k<=n;k++) {  t[k][l]=0.0;  **for** (j=1;j<=n;j++)  t[k][l] += (a[k][j]\*x[j][l]);  printf("%8s %12.6f %12.6f\n"," ",  b[k][l],t[k][l]);  }  }  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  // printf("press RETURN for next problem:\n");  // (void) getchar();  //}  fclose(fp[i]);  }  free\_matrix(t,1,NP,1,MP);  free\_matrix(x,1,NP,1,MP);  free\_matrix(b,1,NP,1,MP);  free\_matrix(u,1,NP,1,NP);  free\_matrix(ai,1,NP,1,NP);  free\_matrix(a,1,NP,1,NP);  **return** 0;  }  #undef NRANSI |

**2.**

**LU Decomposition Method**는 기존 “xlubksb.c” 소스코드를 편집하여 실행하였다. 이 역시 *“lineq1.dat”, “lineq2.dat”, “lineq3.dat”*를 한번에 다 읽어오게 하였고 원래의 **A**-Matrix와 **b**-Vector를 출력하는 부분을 추가하였다. 또한 Decomposition이 진행된 후 **L**-matrix와 **U**-matrix를 출력하게 했고, **U**-Matrix의 Diagonal Elements를 이용하여 A-Matrix의 Determinant를 구하는 것을 추가하였다. 또한 Back-substitution으로 구한 **x**-solution vector를 출력하게 하였고, mprove()를 이용하여 Iterative Improvement를 적용하여 결과값을 출력하여 비교하는 부분 또한 추가하여 주었다.

*편집된< xlubksb.c >, (편집된 부분 중 중요한 부분은 빨간색으로 표시)*

|  |
| --- |
| /\* Driver for routine lubksb \*/  #include <stdio.h>  #include <stdlib.h>  #define NRANSI  #include "nr.h"  #include "nrutil.h"  #define NP 20  #define MAXSTR 80  **int** main(**void**)  {  **int** i,j,k,l,m,n,\*indx;  **float** determ,p,\*x,\*\*a,\*\*b,\*\*c,\*\*xu,\*\*xl,\*bb,\*bbb;  **long** idum=(-13);  //char dummy[MAXSTR];  FILE \*fp[3];  indx=ivector(1,NP);  x=vector(1,NP);  a=matrix(1,NP,1,NP);  b=matrix(1,NP,1,NP);  c=matrix(1,NP,1,NP);  xu=matrix(1, NP, 1, NP);  xl=matrix(1, NP, 1, NP);  bb=vector(1,NP);  bbb=vector(1,NP);    **if** ((fp[0] = fopen("lineq1.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[1] = fopen("lineq2.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[2] = fopen("lineq3.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **for**(i=0;i<3;i++){  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq%d.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",i+1);  //while (!feof(fp)) {  //fgets(dummy,MAXSTR,fp);  //fgets(dummy,MAXSTR,fp);  fscanf(fp[i],"%d ",&n);  fscanf(fp[i],"%d ",&n);  m=1;  //fgets(dummy,MAXSTR,fp);  **for** (k=1;k<=n;k++)  **for** (l=1;l<=n;l++) fscanf(fp[i],"%f ",&a[k][l]);  //fgets(dummy,MAXSTR,fp);  **for** (l=1;l<=m;l++)  **for** (k=1;k<=n;k++) fscanf(fp[i],"%f ",&b[k][l]);  /\* Save matrix a for later testing \*/  **for** (l=1;l<=n;l++)  **for** (k=1;k<=n;k++) c[k][l]=a[k][l];  printf("\nOriginal matrix a : \n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",a[k][l]);  printf("\n");  }  printf("\nOriginal matrix b : \n");  **for** (k=1;k<=n;k++) {  bb[k]=bbb[k] =b[k][1];  printf("%12.6f",b[k][1]);  printf("\n");  }  /\* Do LU decomposition \*/  ludcmp(c,n,indx,&p);  /\* Solve equations for each right-hand vector \*/    /\* Compose separately the lower and upper matrices \*/  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) {  **if** (l > k) {  xu[k][l]=c[k][l];  xl[k][l]=0.0;  } **else** **if** (l < k) {  xu[k][l]=0.0;  xl[k][l]=c[k][l];  } **else** {  xu[k][l]=c[k][l];  xl[k][l]=1.0;  }  }  }  printf("\nlower matrix of the decomposition:\n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",xl[k][l]);  printf("\n");  }  printf("\nupper matrix of the decomposition:\n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",xu[k][l]);  printf("\n");  }  determ = 1.0;  **for**(k=1;k<=n;k++){  determ \*= xu[k][k];  }  printf("\nDeterminant of Matrix a : %12.6f\n",determ);    **for** (k=1;k<=m;k++) {    **for** (l=1;l<=n;l++) x[l]=b[l][k];  lubksb(c,n,indx,x);    printf("\n<Solution vector X> : \n");  **for**(k=1;k<=n;k++){  printf("%12.6f\n",x[k]);  }  printf("\n");    /\* Test results with original matrix \*/  printf("right-hand side vector:\n");  **for** (l=1;l<=n;l++)  printf("%12.6f",b[l][1]);    printf("\n\n%s%s\n","result of matrix applied",  " to sol'n vector");  **for** (l=1;l<=n;l++) {  b[l][k]=0.0;  **for** (j=1;j<=n;j++)  b[l][k] += (a[l][j]\*x[j]);  }  **for** (l=1;l<=n;l++)  printf("%12.6f",b[l][k]);      **for** (l=1;l<=n;l++) x[l] \*= (1.0+0.2\*ran3(&idum));  printf("\n\nSolution vector with noise added:\n");  **for** (l=1;l<=n;l++) printf("%12.6f",x[l]);  printf("\n");  mprove(a,c,n,indx,bb,x);  printf("\nSolution vector recovered by mprove:\n");  **for** (l=1;l<=n;l++) printf("%12.6f",x[l]);  printf("\n");    printf("\n\n%s%s\n","result of matrix applied",  " to sol'n vector with mprove");  **for** (l=1;l<=n;l++) {  bbb[l]=0.0;  **for** (j=1;j<=n;j++)  bbb[l] += (a[l][j]\*x[j]);  }  **for** (l=1;l<=n;l++)  printf("%12.6f",b[l][k]);  printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");  }  //printf("press RETURN for next problem:\n");  //(void) getchar();  //}  fclose(fp[i]);    }  free\_vector(bb,1,NP);  free\_vector(bbb,1,NP);  free\_matrix(c,1,NP,1,NP);  free\_matrix(b,1,NP,1,NP);  free\_matrix(a,1,NP,1,NP);  free\_vector(x,1,NP);  free\_ivector(indx,1,NP);  free\_matrix(xu,1,NP,1,NP);  free\_matrix(xl,1,NP,1,NP);  **return** 0;  }  #undef NRANSI |

**3.**

**Singular Value Decomposition Method**는 기존 “xsvbksb.c”소스코드를 편집하여 실행하였다. 이 역시 *“lineq1.dat”, “lineq2.dat”, “lineq3.dat”*를 한번에 다 읽어오게 하였고 원래의 **A**-Matrix와 **b**-Vector를 출력하는 부분을 추가하였다. 또한 Decomposition이 진행된 후 **U**-matrix, **W**-matrix, **V^T**-matrix 를 출력해주는 부분을 추가하였다.

*편집된< xsvbksb.c >, (편집된 부분 중 중요한 부분은 빨간색으로 표시)*

|  |
| --- |
| /\* Driver for routine svbksb, which calls routine svdcmp \*/  #include <stdio.h>  #include <stdlib.h>  #define NRANSI  #include "nr.h"  #include "nrutil.h"  #define NP 20  #define MP 20  #define MAXSTR 80  **int** main(**void**)  {  **int** i,j,k,l,m,n;  **float** wmax,wmin,\*w,\*x,\*c;  **float** \*\*a,\*\*b,\*\*u,\*\*v;  //char dummy[MAXSTR];  FILE \*fp[3];  w=vector(1,NP);  x=vector(1,NP);  c=vector(1,NP);  a=matrix(1,NP,1,NP);  b=matrix(1,NP,1,MP);  u=matrix(1,NP,1,NP);  v=matrix(1,NP,1,NP);    **if** ((fp[0] = fopen("lineq1.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[1] = fopen("lineq2.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **if** ((fp[2] = fopen("lineq3.dat","r")) == **NULL**)  nrerror("Data file matrx1.dat not found\n");  **for**(i=0;i<3;i++){    printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq%d.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n",i+1);    //while (!feof(fp)) {  //fgets(dummy,MAXSTR,fp);  //fgets(dummy,MAXSTR,fp);  fscanf(fp[i],"%d ",&n);  fscanf(fp[i],"%d ",&n);  m = 1;    //fgets(dummy,MAXSTR,fp);  **for** (k=1;k<=n;k++)  **for** (l=1;l<=n;l++) fscanf(fp[i],"%f ",&a[k][l]);  //fgets(dummy,MAXSTR,fp);  **for** (l=1;l<=m;l++)  **for** (k=1;k<=n;k++) fscanf(fp[i],"%f ",&b[k][l]);  /\* copy a into u \*/  **for** (k=1;k<=n;k++)  **for** (l=1;l<=n;l++)  u[k][l] = a[k][l];    printf("\nOriginal matrix a : \n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++) printf("%12.6f",a[k][l]);  printf("\n");  }    printf("\nOriginal matrix b : \n");  **for** (k=1;k<=n;k++) {  printf("%12.6f",b[k][1]);  printf("\n");  }    /\* decompose matrix a \*/  svdcmp(u,n,n,w,v);    printf("Decomposition matrices:\n");  printf("Matrix u\n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++)  printf("%12.6f",u[k][l]);  printf("\n");  }  printf("Diagonal of matrix w\n");  **for** (k=1;k<=n;k++)  printf("%12.6f",w[k]);  printf("\nMatrix v-transpose\n");  **for** (k=1;k<=n;k++) {  **for** (l=1;l<=n;l++)  printf("%12.6f",v[l][k]);  printf("\n");  }  /\* find maximum singular value \*/  wmax=0.0;  **for** (k=1;k<=n;k++)  **if** (w[k] > wmax) wmax=w[k];  /\* define "small" \*/  wmin=wmax\*(1.0e-6);  /\* zero the "small" singular values \*/  **for** (k=1;k<=n;k++)  **if** (w[k] < wmin) w[k]=0.0;  /\* backsubstitute for each right-hand side vector \*/  **for** (l=1;l<=m;l++) {  printf("\nVector number %2d\n",l);  **for** (k=1;k<=n;k++) c[k]=b[k][l];  svbksb(u,w,v,n,n,c,x);  printf(" solution vector is:\n");  **for** (k=1;k<=n;k++) printf("%12.6f",x[k]);  printf("\n original right-hand side vector:\n");  **for** (k=1;k<=n;k++) printf("%12.6f",c[k]);  printf("\n (matrix)\*(sol'n vector):\n");  **for** (k=1;k<=n;k++) {  c[k]=0.0;  **for** (j=1;j<=n;j++)  c[k] += a[k][j]\*x[j];  }  **for** (k=1;k<=n;k++) printf("%12.6f",c[k]);  printf("\n");  }  printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");  // printf("press RETURN for next problem\n");  // (void) getchar();  // }  fclose(fp[i]);  }  free\_matrix(v,1,NP,1,NP);  free\_matrix(u,1,NP,1,NP);  free\_matrix(b,1,NP,1,MP);  free\_matrix(a,1,NP,1,NP);  free\_vector(c,1,NP);  free\_vector(x,1,NP);  free\_vector(w,1,NP);  **return** 0;  }  #undef NRANSI |

**Results & Discussion:**

*/\*\*출력되는 화면을 사진으로 첨부하려 하였으나 출력되는 양이 너무 많아 한번에 캡쳐하는 것이 불가능하여 ctrl C + V 로 출력 결과를 옮겨서 첨부하였다.(IDE: “Xcode”, Macbook Pro 2015 Retina)\*\*/*

**-01. Gauss-Jordan Elimination Method & Inverse of Matrix “A”**

|  |
| --- |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq1.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **4.000000 2.000000 3.000000 -1.000000**  **-2.000000 -1.000000 -2.000000 2.000000**  **5.000000 3.000000 4.000000 -1.000000**  **11.000000 4.000000 6.000000 1.000000**  **Original matrix b :**  **4.000000**  **-3.000000**  **4.000000**  **11.000000**  **Numerical Recipes run-time error...**  **gaussj: Singular Matrix**  **...now exiting to system...**  **Program ended with exit code: 1** |
| “Lineq1.dat” 에있는 **A**-Matrix는 singular Matrix이기에 해가 무수히 많고 이러한 이유로 default한 “gaussj()”함수로는 해를 구할 수 없고, error를 발생시킨다. |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq2.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **2.000000 -4.000000 -5.000000 5.000000 0.000000**  **-1.000000 1.000000 2.000000 0.000000 4.000000**  **-1.000000 6.000000 0.000000 3.000000 2.000000**  **0.000000 1.000000 3.000000 7.000000 5.000000**  **5.000000 0.000000 8.000000 7.000000 -2.000000**  **Original matrix b :**  **-5.000000**  **2.000000**  **0.000000**  **4.000000**  **-1.000000**  **Inverse of matrix a :**  **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 -0.303963 -0.023201 0.234619 -0.044578**  **0.149114 0.459333 0.051356 -0.171012 0.042492**  **a times a-inverse:**  **1.000000 -0.000000 -0.000000 0.000000 0.000000**  **0.000000 1.000000 0.000000 -0.000000 0.000000**  **0.000000 0.000000 1.000000 -0.000000 0.000000**  **-0.000000 -0.000000 -0.000000 1.000000 -0.000000**  **0.000000 -0.000000 0.000000 -0.000000 1.000000**  **<Solution vector X> :**  **-2.873567**  **-0.612357**  **0.976277**  **0.635819**  **-0.553441**  **Check the following for equality:**  **original matrix\*sol'n**  **vector 1:**  **-5.000000 -5.000000**  **2.000000 2.000000**  **0.000000 -0.000001**  **4.000000 4.000000**  **-1.000000 -1.000002**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq3.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **0.400000 8.200000 6.700000 1.900000 2.200000 5.300000**  **7.800000 8.300000 7.700000 3.300000 1.900000 4.800000**  **5.500000 8.800000 3.000000 1.000000 5.100000 6.400000**  **5.100000 5.100000 3.600000 5.800000 5.700000 4.900000**  **3.500000 2.700000 5.700000 8.200000 9.600000 2.900000**  **3.000000 5.300000 5.600000 3.500000 6.800000 5.700000**  **Original matrix b :**  **-2.900000**  **-8.200000**  **7.700000**  **-1.000000**  **5.700000**  **3.000000**  **Inverse of matrix a :**  **-0.162205 0.122801 0.024068 -0.016431 -0.022840 0.046132**  **0.169407 -0.041117 0.228313 -0.087624 0.180306 -0.395655**  **-0.011636 0.122745 -0.117407 -0.180981 0.015910 0.186766**  **0.105669 -0.051726 -0.108916 0.299774 0.000859 -0.190541**  **-0.053026 -0.042361 0.160508 -0.224034 0.161811 0.015024**  **-0.062341 -0.064694 -0.234216 0.351126 -0.364828 0.434633**  **a times a-inverse:**  **1.000000 0.000000 0.000000 0.000000 0.000000 0.000000**  **0.000000 1.000000 -0.000000 0.000000 0.000000 0.000000**  **-0.000000 0.000000 1.000000 0.000000 0.000000 0.000000**  **0.000000 0.000000 -0.000000 1.000000 0.000000 0.000000**  **0.000000 0.000000 -0.000000 0.000000 1.000000 0.000000**  **0.000000 0.000000 -0.000000 0.000000 0.000000 1.000000**  **<Solution vector X> :**  **-0.326608**  **1.532293**  **-1.044825**  **-1.587447**  **2.928480**  **-2.218931**  **Check the following for equality:**  **original matrix\*sol'n**  **vector 1:**  **-2.900000 -2.900000**  **-8.200000 -8.200001**  **7.700000 7.699999**  **-1.000000 -1.000001**  **5.700000 5.699998**  **3.000000 2.999997**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Program ended with exit code: 0** |
| “Lineq1.dat”를 제외한 나머지들만 실행시켰을 때의 결과이다. 나머지 두 개의 **A**-Matrix는 singular Matrix가 아니기 때문에 gaussj()로 해를 구할 수 있고 **A**의 inverse 또한 존재한다. |

**-02. LU Decomposition Method & Determinant of Matrix “A” & Iterative improvement**

|  |
| --- |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq1.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **4.000000 2.000000 3.000000 -1.000000**  **-2.000000 -1.000000 -2.000000 2.000000**  **5.000000 3.000000 4.000000 -1.000000**  **11.000000 4.000000 6.000000 1.000000**  **Original matrix b :**  **4.000000**  **-3.000000**  **4.000000**  **11.000000**  **lower matrix of the decomposition:**  **1.000000 0.000000 0.000000 0.000000**  **0.454545 1.000000 0.000000 0.000000**  **-0.181818 -0.230769 1.000000 0.000000**  **0.363636 0.461538 -0.375000 1.000000**  **upper matrix of the decomposition:**  **11.000000 4.000000 6.000000 1.000000**  **0.000000 1.181818 1.272727 -1.454545**  **0.000000 0.000000 -0.615385 1.846154**  **0.000000 0.000000 0.000000 0.000000**  **Determinant of Matrix a : -0.000000**  **<Solution vector X> :**  **1.000000**  **-3.000000**  **2.000000**  **0.000000**  **right-hand side vector:**  **4.000000 -3.000000 4.000000 11.000000**  **result of matrix applied to sol'n vector**  **4.000000 -3.000000 4.000000 11.000000**  **Solution vector with noise added:**  **1.093974 -3.252266 2.246411 0.000000**  **Solution vector recovered by mprove:**  **18253927677952.00000036507851161600.000000-54761776742400.000000-18253923483648.000000**  **result of matrix applied to sol'n vector with mprove**  **4.000000 -3.000000 4.000000 11.000000**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq2.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **2.000000 -4.000000 -5.000000 5.000000 0.000000**  **-1.000000 1.000000 2.000000 0.000000 4.000000**  **-1.000000 6.000000 0.000000 3.000000 2.000000**  **0.000000 1.000000 3.000000 7.000000 5.000000**  **5.000000 0.000000 8.000000 7.000000 -2.000000**  **Original matrix b :**  **-5.000000**  **2.000000**  **0.000000**  **4.000000**  **-1.000000**  **lower matrix of the decomposition:**  **1.000000 0.000000 0.000000 0.000000 0.000000**  **-0.200000 1.000000 0.000000 0.000000 0.000000**  **0.400000 -0.666667 1.000000 0.000000 0.000000**  **0.000000 0.166667 -0.383178 1.000000 0.000000**  **-0.200000 0.166667 -0.467290 0.372304 1.000000**  **upper matrix of the decomposition:**  **5.000000 0.000000 8.000000 7.000000 -2.000000**  **0.000000 6.000000 1.600000 4.400000 1.600000**  **0.000000 0.000000 -7.133333 5.133333 1.866667**  **0.000000 0.000000 0.000000 8.233644 5.448598**  **0.000000 0.000000 0.000000 0.000000 2.177071**  **Determinant of Matrix a : -3835.999512**  **<Solution vector X> :**  **-2.873566**  **-0.612357**  **0.976277**  **0.635819**  **-0.553441**  **right-hand side vector:**  **-5.000000 2.000000 0.000000 4.000000 -1.000000**  **result of matrix applied to sol'n vector**  **-4.999999 2.000000 -0.000000 4.000001 -1.000000**  **Solution vector with noise added:**  **-3.023735 -0.634681 1.029285 0.742942 -0.557133**  **Solution vector recovered by mprove:**  **-2.873566 -0.612357 0.976277 0.635818 -0.553441**  **result of matrix applied to sol'n vector with mprove**  **-4.999999 2.000000 -0.000000 4.000001 -1.000000**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq3.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **0.400000 8.200000 6.700000 1.900000 2.200000 5.300000**  **7.800000 8.300000 7.700000 3.300000 1.900000 4.800000**  **5.500000 8.800000 3.000000 1.000000 5.100000 6.400000**  **5.100000 5.100000 3.600000 5.800000 5.700000 4.900000**  **3.500000 2.700000 5.700000 8.200000 9.600000 2.900000**  **3.000000 5.300000 5.600000 3.500000 6.800000 5.700000**  **Original matrix b :**  **-2.900000**  **-8.200000**  **7.700000**  **-1.000000**  **5.700000**  **3.000000**  **lower matrix of the decomposition:**  **1.000000 0.000000 0.000000 0.000000 0.000000 0.000000**  **0.051282 1.000000 0.000000 0.000000 0.000000 0.000000**  **0.705128 0.379123 1.000000 0.000000 0.000000 0.000000**  **0.653846 -0.042051 0.242635 1.000000 0.000000 0.000000**  **0.384615 0.271108 -0.192761 0.328693 1.000000 0.000000**  **0.448718 -0.131761 -0.638113 1.354025 1.191337 1.000000**  **upper matrix of the decomposition:**  **7.800000 8.300000 7.700000 3.300000 1.900000 4.800000**  **0.000000 7.774359 6.305128 1.730769 2.102564 5.053846**  **0.000000 0.000000 -4.819904 -1.983097 2.963126 1.099357**  **0.000000 0.000000 0.000000 4.196258 3.827151 1.707318**  **0.000000 0.000000 0.000000 0.000000 4.812427 2.134437**  **0.000000 0.000000 0.000000 0.000000 0.000000 -2.741017**  **Determinant of Matrix a : 16178.401367**  **<Solution vector X> :**  **-0.326608**  **1.532292**  **-1.044826**  **-1.587447**  **2.928480**  **-2.218930**  **right-hand side vector:**  **-2.900000 -8.200000 7.700000 -1.000000 5.700000 3.000000**  **result of matrix applied to sol'n vector**  **-2.900000 -8.199999 7.700000 -0.999998 5.699999 2.999999**  **Solution vector with noise added:**  **-0.357722 1.739582 -1.060572 -1.603810 3.023567 -2.316363**  **Solution vector recovered by mprove:**  **-0.326608 1.532293 -1.044825 -1.587448 2.928480 -2.218931**  **result of matrix applied to sol'n vector with mprove**  **-2.900000 -8.199999 7.700000 -0.999998 5.699999 2.999999**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Program ended with exit code: 0** |
| “lineq1.dat”의 경우, **A**-matrix가 singular 이기 때문에 **U**-Matrix의 diagonal elements중 0이 존재하고 이때문에 Determinant가 0이 된다. 다른 data의 경우 모두 unique한 Solution을 갖는다. 각각 출력되는 solution을 위의 “xgaussj.c”의 solution과 비교했을 때 오차범위 안에서 같은 값을 출력하는 것을 확인할 수 있다.  또한 **L**-matrix와 **U**-matrix를 출력된 상태 그대로 서로 Matrix Multiplication을 해줬을 때 원래의 **A**-Matrix와 완전히 같지는 않고 **A**-matrix에서 각 행들의 위치가 바뀌어 있는 상태의 matrix가 되는데, 각 행의 위치를 바꾸는 횟수를 잘 조절하여 Determinant값에 변화를 주지 않는 선에서 각각 바뀐 행의 위치와 똑같이 **b**-vector도 위치를 바꿔준 다음 Back-Substitution을 하면 본래의 **x**-solution vector 와 같는 solution을 구할 수 있는 것을 확인 할 수 있다.  또한 mprove()를 사용할 때, 임의로 **x**-solution vector에 noise값을 추가하여 준 다음 mprove()를 실행해봤는데 오차범위 내에서 LU-Back substitution 을 noise 없이 했을 때랑 오차범위내에서 같은 값을 출력하는 것을 확인 할 수 있었다. |

**-03. Singular Value Decomposition Method**

|  |
| --- |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq1.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **4.000000 2.000000 3.000000 -1.000000**  **-2.000000 -1.000000 -2.000000 2.000000**  **5.000000 3.000000 4.000000 -1.000000**  **11.000000 4.000000 6.000000 1.000000**  **Original matrix b :**  **4.000000**  **-3.000000**  **4.000000**  **11.000000**  **Decomposition matrices:**  **Matrix u**  **-0.334681 -0.341889 0.004004 0.878114**  **0.184702 0.672408 0.636647 0.329292**  **-0.436250 -0.410160 0.730082 -0.329293**  **-0.814591 0.512590 -0.248283 -0.109764**  **Diagonal of matrix w**  **16.084492 2.956074 0.742099 0.000000**  **Matrix v-transpose**  **-0.798899 -0.337044 -0.497746 0.020252**  **0.296108 -0.181425 -0.316495 0.882743**  **-0.455427 0.766042 0.228209 0.392030**  **-0.258199 -0.516398 0.774597 0.258199**  **Vector number 1**  **solution vector is:**  **1.733333 -1.533334 -0.200000 -0.733334**  **original right-hand side vector:**  **4.000000 -3.000000 4.000000 11.000000**  **(matrix)\*(sol'n vector):**  **4.000001 -3.000001 4.000001 11.000001**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq2.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **2.000000 -4.000000 -5.000000 5.000000 0.000000**  **-1.000000 1.000000 2.000000 0.000000 4.000000**  **-1.000000 6.000000 0.000000 3.000000 2.000000**  **0.000000 1.000000 3.000000 7.000000 5.000000**  **5.000000 0.000000 8.000000 7.000000 -2.000000**  **Original matrix b :**  **-5.000000**  **2.000000**  **0.000000**  **4.000000**  **-1.000000**  **Decomposition matrices:**  **Matrix u**  **-0.071261 -0.309980 -0.100152 0.695179 0.636812**  **-0.116744 -0.738275 0.534932 -0.374887 0.120944**  **-0.220640 -0.196165 -0.745625 -0.506976 0.316001**  **-0.569114 0.531849 0.363562 -0.199649 0.470328**  **-0.780205 -0.193696 -0.125232 0.281605 -0.508702**  **Diagonal of matrix w**  **13.865567 0.777249 4.662140 9.169546 8.326208**  **Matrix v-transpose**  **-0.267293 -0.124384 -0.564432 -0.754636 -0.158192**  **-0.841428 -0.184618 0.153527 0.294218 -0.384422**  **-0.132079 -0.680942 0.355943 -0.229363 0.582727**  **0.401355 -0.697647 -0.280470 0.275767 -0.444401**  **-0.204995 -0.007203 -0.672672 0.464010 0.538639**  **Vector number 1**  **solution vector is:**  **-2.873566 -0.612357 0.976278 0.635819 -0.553441**  **original right-hand side vector:**  **-5.000000 2.000000 0.000000 4.000000 -1.000000**  **(matrix)\*(sol'n vector):**  **-4.999998 1.999999 -0.000000 4.000002 -0.999994**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **\*\*\*\*\*\*\*\*\*\*\*\*\*<lineq3.dat> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Original matrix a :**  **0.400000 8.200000 6.700000 1.900000 2.200000 5.300000**  **7.800000 8.300000 7.700000 3.300000 1.900000 4.800000**  **5.500000 8.800000 3.000000 1.000000 5.100000 6.400000**  **5.100000 5.100000 3.600000 5.800000 5.700000 4.900000**  **3.500000 2.700000 5.700000 8.200000 9.600000 2.900000**  **3.000000 5.300000 5.600000 3.500000 6.800000 5.700000**  **Original matrix b :**  **-2.900000**  **-8.200000**  **7.700000**  **-1.000000**  **5.700000**  **3.000000**  **Decomposition matrices:**  **Matrix u**  **-0.353729 0.373714 0.751240 0.209290 0.322161 0.152517**  **-0.457606 0.385934 -0.478084 0.601435 -0.226633 0.001283**  **-0.414094 0.321511 -0.200360 -0.714971 -0.088791 0.407363**  **-0.394852 -0.207079 -0.280703 -0.113860 0.732258 -0.416233**  **-0.418250 -0.741479 0.073238 0.173781 -0.108773 0.477368**  **-0.403927 -0.123898 0.287702 -0.200315 -0.537524 -0.640048**  **Diagonal of matrix w**  **30.634525 9.954135 5.350266 4.867327 1.819682 1.119591**  **Matrix v-transpose**  **-0.348552 -0.510097 -0.430995 -0.317609 -0.416918 -0.397315**  **0.090927 0.540705 0.077793 -0.543461 -0.597330 0.202895**  **-0.905129 0.134551 0.330646 -0.069389 0.146158 0.164723**  **0.055301 -0.155486 0.687702 0.355621 -0.490215 -0.364766**  **-0.212118 0.313933 -0.465455 0.686520 -0.384793 0.142930**  **-0.054152 0.553720 -0.096350 -0.034405 0.244206 -0.787628**  **Vector number 1**  **solution vector is:**  **-0.326609 1.532292 -1.044825 -1.587447 2.928479 -2.218929**  **original right-hand side vector:**  **-2.900000 -8.200000 7.700000 -1.000000 5.700000 3.000000**  **(matrix)\*(sol'n vector):**  **-2.899993 -8.200000 7.699998 -1.000001 5.700000 3.000003**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Program ended with exit code: 0** |
| “lineq1.dat”의 경우, **A**-matrix가 singular 이기 때문에 **W**-Matrix의 diagonal elements중 0이 존재하고 이때문에 Determinant가 0이 된다. 이때 출력된 solution은 위의 “xlubksb.c”에서와 다른 값을 갖는데 이는 SVD특성상 **W**-matrix의 Inverse를 구할 때 1/0을 0으로 놓고 계산하기 때문에 해를 근사 하여 구하게 되기 때문이다. 다른 data의 경우 모두 unique한 Solution을 갖는다. 각각 출력되는 solution을 위의 “xgaussj.c”의 solution과 비교했을 때 오차범위 안에서 같은 값을 출력하는 것을 확인할 수 있다. |

**-Overall Discussion**

|  |
| --- |
| **-Gauss-Jordan Elimination**  이 Method는 Gauss-reduction을 사용하여 solution을 구하는 방식으로 Inverse matrix를 쉽게 구할 수 있는 방식이다. 하지만 inverse matrix가 존재하지 않는 singular matrix인 경우 solution을 구할 수 없고, 구해야 하는 Solution 이 계속해서 바뀌는 경우 (ex: **b**-vector가 계속해서 바뀌는 경우) Solution을 구하는 속도가 매우 느리다는 단점이 있다.  **-LU Decomposition**  이 Method는 **A**-matrix를 Lower-triangular Matrix와 Upper-triangular Matrix의 곱으로 변환시킨 다음 Back substitution을 통해 solution을 구하는 방식이다. 이 방식은 처음에 **L**-matrix와 **U**-matrix를 만드는데 오래 걸리긴 하지만 한번 **LU decomposition**을 해놓면 위에 언급한 상황처럼 구해야 하는 solution이 계속해서 바뀌는 경우 Back substitution만 활용하여 빠른 시간에 여러가지 solution을 구할 수 있다는 장점이 있다. 하지만 이 역시 singular matrix인 경우에는 정확한 해가 아닌 위에 출력 결과에 나온 것처럼 하나의 unknown을 0으로 단정하고 나머지 solution을 구하기 때문에 singular matrix에는 취약할 수 있다.  -**Singular Value Decomposition**  이 Method는 **A**-matrix를 **U**-matrix, **W**-matrix, **V^T**-Matrix의 곱으로 변환시킨 다음, 이들의 inverse를 이용하여 solution을 구하는 방식이다. **U**-Matrix와 **V-**Matrix는 Orthonormal 하기에 자신의 Transpose가 자신의 Inverse인 성질을 이용하여 inverse를 쉽게 구할 수 있고, **W**-matrix의 경우 diagonal matrix이므로 diagonal elements에 역수를 취해주면 쉽게 inverse를 구할 수 있다. 만약 **A**-matrix가 singular matrix인 경우 **W**의 diagonal elements중 0이 존재하게 되고 이런 경우 **W**의 inverse를 구할 때 1/0을 0으로 처리해 주어 solution에 가장 근사한 값으로 solution을 구하게 된다. Singular matrix의 solution을 구할 때 가장 좋은 방법이라 할 수 있겠다. |