# Assignment 3

**2024017201 컴퓨터소프트웨어학부 최선웅**
**12034 수치해석**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "nrutil.h"

#define NRANSI
#include "nr.h"
#undef fmin

void read_system(const char *fname, int *n, float ***A, float **b) {
   FILE *fp = fopen(fname, "r");
   if (!fp) { perror(fname); exit(1); }
   fscanf(fp, "%d %d", n, n);
   *A = matrix(1, *n, 1, *n);
   *b = vector(1, *n);
   for (int i = 1; i <= *n; i++)
      for (int j = 1; j <= *n; j++) {
         double tmp; fscanf(fp, "%lf", &tmp);
         (*A)[i][j] = (float)tmp;
      }
   for (int i = 1; i <= *n; i++) {
      double tmp; fscanf(fp, "%lf", &tmp);
      (*b)[i] = (float)tmp;
   }
   fclose(fp);
}

void compute_inverse(float **Aorig, int n, float **invA) {
   int *indx = ivector(1, n);
   float d;
   float **A = matrix(1, n, 1, n);
   for (int i=1;i<=n;i++) for(int j=1;j<=n;j++) A[i][j]=Aorig[i][j];
```

```
    ludcmp(A, n, indx, &d);
    for (int j=1;j<=n;j++) {
        float *col = vector(1, n);
        for (int i=1;i<=n;i++) col[i]=0.0f;
        col[j]=1.0f;
        lubksb(A, n, indx, col);
        for (int i=1;i<=n;i++) invA[i][j]=col[i];
        free_vector(col,1,n);
    }
    free_ivector(indx,1,n);
    free_matrix(A,1,n,1,n);
}

float compute_determinant(float **Aorig, int n) {
    int *indx = ivector(1, n);
    float d;
    float **A = matrix(1, n, 1, n);
    for (int i=1;i<=n;i++) for(int j=1;j<=n;j++) A[i][j]=Aorig[i][j];
    ludcmp(A, n, indx, &d);
    float det = d;
    for (int i=1;i<=n;i++) det *= A[i][i];
    free_ivector(indx,1,n);
    free_matrix(A,1,n,1,n);
    return det;
}

void compute_residual(float **A, float *b, float *x, int n, float *r) {
    for(int i=1;i<=n;i++){
        r[i]=b[i];
        for(int j=1;j<=n;j++) r[i]-=A[i][j]*x[j];
    }
}

float vector_norm(float *v, int n){
    float s=0; for(int i=1;i<=n;i++) s+=v[i]*v[i];
    return sqrtf(s);
}
```

```
int main(void) {
    char fname[32];
    for (int idx = 1; idx <= 3; idx++) {
        int n;
        float **A, *b;
        sprintf(fname, "lineq%d.dat", idx);
        read_system(fname, &n, &A, &b);

        printf("=== System %d (n=%d) ===\n", idx, n);

        float det = compute_determinant(A, n);

        if (fabsf(det) < 1e-7f) {
            printf("Gauss-Jordan Elimination:\nMatrix is singular (det=%e), skip
ping\n", det);
        } else {
            float **Aj = matrix(1, n, 1, n);
            float **Bj = matrix(1, n, 1, 1);
            for(int i=1;i<=n;i++){
                for(int j2=1;j2<=n;j2++) Aj[i][j2]=A[i][j2];
                Bj[i][1]=b[i];
            }
            gaussj(Aj, n, Bj, 1);
            printf("Gauss-Jordan Elimination:\n");
            for(int i=1;i<=n;i++) printf("%.6f\t\t", Bj[i][1]);
            printf("\n");
            free_matrix(Aj,1,n,1,n);
            free_matrix(Bj,1,n,1,1);
        }

        float **Alu = matrix(1,n,1,n);
        float *xlu = vector(1,n);
        int *indx = ivector(1,n);
        for(int i=1;i<=n;i++){
            for(int j2=1;j2<=n;j2++) Alu[i][j2]=A[i][j2];
            xlu[i]=b[i];
        }
        float dsign;
```

```
ludcmp(Alu, n, indx, &dsign);
lubksb(Alu, n, indx, xlu);
printf("LU Decomposition:\n");
for(int i=1;i<=n;i++) printf("%.6f\t\t", xlu[i]);
printf("\n");

float **Asvd = matrix(1,n,1,n);
float *w = vector(1,n);
float **V = matrix(1,n,1,n);
float *xsvd = vector(1,n);
for(int i=1;i<=n;i++){
   for(int j2=1;j2<=n;j2++) Asvd[i][j2]=A[i][j2];
   xsvd[i]=0.0f;
}
svdcmp(Asvd, n, n, w, V);
float wmax=0, wmin=1e30f;
for(int j2=1;j2<=n;j2++){
   if(w[j2]>wmax) wmax=w[j2];
   if(w[j2]>0 && w[j2]<wmin) wmin=w[j2];
}
for(int j2=1;j2<=n;j2++){
   float sum=0;
   if(w[j2]>wmax*1e-6f){
      for(int i=1;i<=n;i++) sum += Asvd[i][j2]*b[i];
      sum /= w[j2];
   }
   for(int i=1;i<=n;i++) xsvd[i]+=V[i][j2]*sum;
}
printf("SV Decomposition:\n");
for(int i=1;i<=n;i++) printf("%.6f\t\t", xsvd[i]);
printf("\n");

float *rlu = vector(1,n);
for(int it=1; it<=3; it++){
   mprove(A, Alu, n, indx, b, xlu);
}
printf("Iterative Improvement:\n");
for(int i=1;i<=n;i++) printf("%.6f\t\t", xlu[i]);
```

```
        printf("\n");
        free_vector(rlu,1,n);

        float **invA = matrix(1,n,1,n);
        compute_inverse(A,n,invA);
        printf("Inverse Matrix:\n");
        for(int i=1;i<=n;i++){
            for(int j2=1;j2<=n;j2++){
                printf("%.5f ", invA[i][j2]);
            }
            printf("\n");
        }

        printf("Determinant:\n");
        printf("%.6f\n", det);

        free_matrix(A,1,n,1,n);
        free_vector(b,1,n);
        free_matrix(Alu,1,n,1,n);
        free_vector(xlu,1,n);
        free_ivector(indx,1,n);
        free_matrix(Asvd,1,n,1,n);
        free_vector(w,1,n);
        free_matrix(V,1,n,1,n);
        free_vector(xsvd,1,n);
        free_matrix(invA,1,n,1,n);

        printf("\n");
    }
    return 0;
}
```

## 전체 목적

3개의 선형 연립방정식 시스템($\boldsymbol{Ax} = \boldsymbol{b}$)을 서로 다른 수치해석적 방법으로 풀고 비교하는 코드이다.

## 메인 프로그램 흐름

각 시스템(lineq1.dat, lineq2.dat, lineq3.dat)에 대해 순차적으로:

1. 시스템 정보 출력: `=== System i (n = 차수) ===`
2. 4가지 수치해석적 방법으로 연립방정식 해결:
    - Gauss-Jordan 소거법: `gaussj()` 사용(특이행렬 검사 포함)
    - LU 분해법: `ludcmp()` + `lubksb()` 사용
    - SVD: `svdcmp()` 사용
    - 반복 개선법: `mprove()` 를 3회 적용하여 LU 해 개선
3. 행렬 분석:
    - 역행렬 계산 및 출력
    - 행렬식 계산 및 출력

## 핵심 특징

- **Numerical Recipes** 라이브러리 사용 (1-based 인덱싱)
- 메모리 관리: 각 방법마다 독립적인 메모리 할당/해제
- 에러 처리: 파일 읽기 실패, 특이행렬 상황 처리
- 비교 분석: 동일한 시스템을 여러 방법으로 풀어 정확도/안정성 비교 가능

## 결과

결과는 다음과 같다.

### lieq1.dat

```
=== System 1 (n=4) ===
Gauss-Jordan Elimination:
Matrix is singular (det=-7.999999e-20), skipping
LU Decomposition:
1.000000                -3.000000               2.000000                0.000000
SV Decomposition:
1.733333                -1.533333               -0.200000               -0.733333
Iterative Improvement:
1.000000                -3.000000               2.000000                0.000000
Inverse Matrix:
-100000002004087734272.00000 -37500000751532900352.00000 37500000751532900352.00000 12500001350022594560.00000
-200000056784733601792.00000 -75000019095251845120.00000 75000010299158822912.00000 25000004899068444672.00000
300000041196635291648.00000 112500015448738234368.00000 -112500006652645212160.00000 -37500005149579411456.00000
100000002004087734272.00000 37500000751532900352.00000 -37499996353486389248.00000 -12500001350022594560.00000
Determinant:
-0.000000
```

`lieq1.dat` 의 $\boldsymbol{A}$행렬이 singular하여 위와 같은 결과가 발생하였다.

## lieq2.dat

```
=== System 2 (n=5) ===
Gauss-Jordan Elimination:
-2.873567               -0.612357               0.976277               0.635819               -0.553441
LU Decomposition:
-2.873566               -0.612357               0.976277               0.635819               -0.553441
SV Decomposition:
-2.873570               -0.612358               0.976278               0.635820               -0.553443
Iterative Improvement:
-2.873566               -0.612357               0.976277               0.635818               -0.553441
Inverse Matrix:
0.35454 0.76694 0.20777 -0.59541 0.25313
0.03545 0.12669 0.19578 -0.15954 0.05031
-0.13869 -0.09854 -0.09672 0.12409 0.01642
-0.05214 -0.30396 -0.02320 0.23462 -0.04458
0.14911 0.45933 0.05136 -0.17101 0.04249
Determinant:
3835.999512
```

## lieq3.dat

```
=== System 3 (n=6) ===
Gauss-Jordan Elimination:
-0.326608               1.532293               -1.044825               -1.587447               2.928480               -2.218931
LU Decomposition:
-0.326608               1.532292               -1.044826               -1.587447               2.928480               -2.218930
SV Decomposition:
-0.326608               1.532292               -1.044825               -1.587448               2.928480               -2.218930
Iterative Improvement:
-0.326608               1.532293               -1.044826               -1.587448               2.928480               -2.218931
Inverse Matrix:
-0.16221 0.12280 0.02407 -0.01643 -0.02284 0.04613
0.16941 -0.04112 0.22831 -0.08762 0.18031 -0.39565
-0.01164 0.12274 -0.11741 -0.18098 0.01591 0.18677
0.10567 -0.05173 -0.10892 0.29977 0.00086 -0.19054
-0.05303 -0.04236 0.16051 -0.22403 0.16181 0.01502
-0.06234 -0.06469 -0.23422 0.35113 -0.36483 0.43463
Determinant:
16178.401367
```

`lineq2.dat` , `lineq3.dat` 의 경우에는 모든 method들이 공통적으로 오차 범위 내에서 같은 해를 도출하는 것을 확인 할 수 있다.