Homework#5

2015004302 곽상원

목차

1. Purpose of program
2. Experimental process
3. Result
4. Discussion

---

1. Purpose of program

선형방정식 Ax=b(A는 NxN matrix)에 대해, 여러 가지 method를 이용해 **해와 A의 Inverse matrix, Determinant**를 구하는 것이 목적이다.

▶**Gauss-Jordan Elimination**: Augmented matrix [A:x]에 대해, 각 행을 normalize 후, partial pivoting을 이용해 제일 앞쪽 성분을 제거해나가면서 upper triangular 꼴로 바꾼다. 이후 back substitution을 통해 x를 구한다.

▶**LU Decomposition**: A를 lower, upper triangular matrix 곱으로 나타낸다, Ax=b -> LUx=b 이때, Ux=c라고 하면 Lc=b에서 back substitution로 c를 구하고 Ux=c에서 substitution으로 x를 구한다.

▶**SVD**: $A = UWV^T$(U는 , W는 대각행렬,V는) 꼴로 decomposition한 후에 여러 차례 substitution을 통해 x를 구한다.

▶**Iterative Improvement**: direct한 방법으로 x를 구하다보면 어쩔 수 없이 연산이 많아져서 roundoff error가 증폭된다. 이를 최대한 제거해서 오차가 적은 x를 구하기위한 방법이다.

2 .Experimental process

-Ax=b에서 A, b에 대한 입력은 텍스트 파일을 이용했다. 각 텍스트 파일의 첫 번째 줄은 A의 행과 열에 대한 정보, 그 후 A와 b의 값이 입력되어있다.**(lineq1.dat, lineq2.dat, lineq3.dat)**

❶Gauss-Jordan Elimination, LU Decomposition, SVD를 이용해 x구하기

▶*Gauss-Jordan Elimination(gaussj.c)*

<code>

```
//Ax=b를 만족하는 x구하기
		//1.gauss-jordan elimination

		printf("<Gauss-Jordan elimination>\n");
		sin = 1;
		//a set of solution은 b_gauss에 저장된다.
		gaussj(a, n, b_gauss,1, &sin);
		//sin이 1이면 non singular
		if (sin == 1) {
			printf("solution(x) : ");
			for (int r = 1; r <= n; r++) printf("%9f ", b_gauss[r][1]);
		}
```

▶*LU Decomposition(ludcmp.c & lubksb.c)*

<code>

```
//2.LU decomposition
		printf("\n<LU Decomposition>\n");
		sin = 1;
		ludcmp(a_lu, n, indx, &d);
		lubksb(a_lu, n, indx, b_lu,&sin);
		if (sin == 1) {
			printf("solution(x) : ");
			for (int r = 1; r <= n; r++) printf("%9f ", b_lu[r]);
		}
```

-ludcmp로 LU로 분해된 matrix와 lubksb(back substitution함수)를 이용해 x를 구한다.

-기존의 gaussj와 ludcmp은 singular matrix를 입력으로 받아들였을 때, 프로그램을 종료하도록 설계되어있었다. 이를 함수만 종료하고 sin이라는 변수로 singular여부를 알 수 있도록 수정했다.

▶ *SVD(svdcmp.c & svbksb.c)*

<code>

```
svdcmp(u,n,n,w,v);
                /* 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 */
                svbksb(u, w, v, n, n, b, x);
```

-svdcmp에 의해 $A = UWV^T$로 분해된 matrix와 svbksb(back substitution함수)를 이용해 x를 구한다.
-위의 두 가지 method와 다른 main함수 내에서 실행했다.(xsvbksb.c)

❷Iterative Improvement를 이용해 x구하기(mprove.c)

<code>

```
printf("<iterative improvement>\n");
                sin = 1;
                ludcmp(a_lud, n, indx, &d);
                mprove(a, a_lud, n, indx, b, x, &sin);
                //singular 여부확인
                if (sin == 1) {
                        printf("solution(x) : ");
                        for (int r = 1; r <= n; r++) printf("%f ", x[r]);
                }
```

-Ax=b에서 해라고 추측되는 $\underline{x}$를 입력으로 집어넣고 $\underline{b}$를 얻는다. 원래 값인 b를 알기 때문에 ludcmp을 이용해, 참 x, 참b에 대해 A(x-$\underline{x}$) = b-$\underline{b}$에서 x의 오차를 알아내고 참 x를 얻는다.

❸A의 Inverse matrix, Determinant구하기

▶*Inverse matrix(func.c의 inverse())*

<code>

```
//한 열씩 inverse matrix를 구하는 함수
void inverse(float **a, int n, int *indx, float **a_inverse) {
	int i, j,temp;
	float* col = (float*)malloc(sizeof(float)*(n+1));

	for (j= 1 ; j <= n; j++) {
		for (i = 1; i <= n; i++) col[i] = 0.0;
		//identity matrix의 한 열씩 가져온다
		col[j] = 1.0;
		lubksb(a, n, indx, col,&temp);
		for (i = 1; i <= n; i++) a_inverse[i][j] = col[i];
	}
	free(col);
}
```

▶*Determinant*

<code>

```
ludcmp(a, n, indx, &d);
		//determinant 구하기
		for (int i = 1; i <= n; i++) d *= a[i][i];
		printf("Determinant : %f\n", d);
```

-ludcmp의 결과로 나온 matrix가 triangular matrix이므로 대각성분을 쭉 곱해 값을 얻는다.

3.Result

▶*입력*

<lineq1.dat>

4	4		
4	2	3	-1
-2	-1	-2	2
5	3	4	-1
11	4	6	1
4	-3	4	11

<lineq2.dat>

5	5			
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
-5	2	0	4	-1

<lineq3.dat>

6 6
0.4 8.2 6.7 1.9 2.2 5.3
7.8 8.3 7.7 3.3 1.9 4.8
5.5 8.8 3.0 1.0 5.1 6.4
5.1 5.1 3.6 5.8 5.7 4.9
3.5 2.7 5.7 8.2 9.6 2.9
3.0 5.3 5.6 3.5 6.8 5.7
-2.9 -8.2 7.7 -1.0 5.7 3.0

▶*결과*

➊Gauss-Jordan Elimination, LU Decomposition, SVD

Gauss-Jordan & LUD

```
-------lineq1.dat-------
<Gauss-Jordan elimination>
gassuj: Singular Matrix

<LU Decomposition>
lubksb: Singular Matrix


-------lineq2.dat-------
<Gauss-Jordan elimination>
solution(x) : -2.873567 -0.612357  0.976277  0.635819 -0.553441
<LU Decomposition>
solution(x) : -2.873566 -0.612357  0.976277  0.635819 -0.553441

-------lineq3.dat-------
<Gauss-Jordan elimination>
solution(x) : -0.326608  1.532293 -1.044825 -1.587447  2.928480 -2.218931
<LU Decomposition>
solution(x) : -0.326608  1.532292 -1.044826 -1.587447  2.928480 -2.218930
```

SVD

```
-------lineq1.dat-------
 solution vector is:
   1.733333   -1.533334   -0.200000   -0.733334

-------lineq2.dat-------
 solution vector is:
  -2.873566   -0.612357    0.976278    0.635819   -0.553441

-------lineq3.dat-------
 solution vector is:
  -0.326609    1.532292   -1.044825   -1.587447    2.928479   -2.218929
```

❷Iterative Improvement

```
-------lineq1.dat-------
<iterative improvement>
lubksb: Singular Matrix



-------lineq2.dat-------
<iterative improvement>
solution(x) : -2.873566 -0.612357 0.976277 0.635818 -0.553441


-------lineq3.dat-------
<iterative improvement>
solution(x) : -0.326609 1.532294 -1.044827 -1.587448 2.928481 -2.218932
```

❸A의 Inverse matrix, Determinant

```
-------lineq1.dat-------
Determinant : -0.000000
Inverse doesn't exist


-------lineq2.dat-------
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


-------lineq3.dat-------
Determinant : 16178.401367
Inverse matrix
 -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  -0.042362   0.160508  -0.224034   0.161811   0.015024
 -0.062341  -0.064694  -0.234216   0.351126  -0.364828   0.434633
```

4.Discussion

▶*Gauss-Jordan Elimination, LU Decomposition, SVD 장점, 단점*

-Gauss-Jordan은 바로 reduction을 통해 해를 찾을 수 있지만, LUD와 SVD는 A를 decomposition해야 한다. 그러다보니 해를 한번만 찾을 때는 Gauss-Jordan이 나머지 2개의 방법보다 빠르다. 하지만 Ax=b에서 b가 바뀌면서 여러 번 해를 찾아야할 경우 LUD, SVD는 이미 분해해놓은 matrixes에 대해 substitution만 진행하면 되므로 훨씬 속도가 빠를 것이다.

-SVD는 나머지 2개의 방법에 비해 Singular matrix인 경우에도 해를 구할 수 있다. 물론 근사 해를 구하는 것이지만, 현실세계에서는 항상 딱 떨어지는 해를 구할 수 없기 때문에 이는 큰 강점이 된다.

▶*Iterative Improvement결과분석*

-이 방법은 machine precision을 온전히 다 쓰고 싶어 사용되는 방법이다. 물론 추측되는 값 x를 넣고, 이 방법 자체로도 해를 구할 수 있었다. 위의 결과는 1회 실행한 결과로, 이 자체만으로는 별 효과를 기대할 수 없다. 위의 Gauss-Jordan Elimination, LU Decomposition으로 먼저 해를 구하고, 이 해를 Iterative Improvement를 거듭 실행하면 machine의 precision를 충분히 다 이용해서 오차가 굉장히 작은 해를 구할 수 있을 것이라 기대된다.