**Problem 2**

function [one\_norm, two\_norm, max\_norm] = mynorm(A)

for i = 1:size(A,1)

    row\_sum(i) = sum(abs(A(i,:)));

end

for j = 1:size(A,2)

    col\_sum(i) = sum(abs(A(:,j)));

end

C = eig(A'\*A);

C = C.^0.5;

one\_norm = max(col\_sum);

two\_norm = max(C);

max\_norm = max(row\_sum);

end

>> A = randn(2000,2000);

>> [o t m] = mynorm(A)

o =

1.5672e+03

t =

89.0632

m =

1.6826e+03

>> [norm(A,1) norm(A,2) norm(A,’inf’)]

ans =

[1692.4 89.1 1682.6]

%Expnanation:

Since the 1-norm and max-norm are simply summation of entries of the matrix, small perturbation will not have great affect on the norm. However, if the matrix is big and bad conditioned, small perturbation can change the eigenvalue a lot. However, using the implementation, I don’t see the dimension will pose significant problem. I’ve tried several random matrix and all work well.

**Problem 7.14**

function A = matgen(n, condno)

[U,R] = qr(randn(n,n));

[V,R] = qr(randn(n,n));

Sigma = zeros(n,n);

for i=1:n

    Sigma(i,i) = condno^(-(i-1)/(n-1));

end

A = U\*Sigma\*V';

%changing value of n

n = 5;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Gs\_err | Gs\_res | Inv\_err | Inv\_res | Cra\_err | Cra\_res |
| 1 | 1e-16 | 1.4919 | 1.3e-16 | 1.4919 | 2.6e-16 | 1.95 |
| 10^4 | 5e-14 | 0.7302 | 4e-14 | 0.7302 | 1e-13 | 1.48 |
| 10^8 | 9e-10 | 0.9061 | 2.3e-9 | 0.9061 | 2e-9 | 1.13 |
| 10^12 | 1.9e-5 | 1.3585 | 1.4e-5 | 1.3585 | 7e-6 | 0.069 |
| 10^16 | 7.2e-3 | 0.9513 | 0.0167 | 0.9599 | 0.1276 | 1.4867 |

%Gaussian

A1 = matgen(n,1);

A2 = matgen(n,10^4);

A3 = matgen(n,10^8);

A4 = matgen(n,10^12);

A5 = matgen(n,10^16);

xtrue = randn(n,1);

b1 = A1\*xtrue;

b2 = A2\*xtrue;

b3 = A3\*xtrue;

b4 = A4\*xtrue;

b5 = A5\*xtrue;

x1 = A1\b1;

x2 = A2\b2;

x3 = A3\b3;

x4 = A4\b4;

x5 = A5\b5;

rel\_err(1) = norm(x1-xtrue)/norm(xtrue);

rel\_res(1) = norm(b1) - norm(A1\*x1)/(norm(A1)\*norm(x1));

rel\_err(2) = norm(x2-xtrue)/norm(xtrue);

rel\_res(2) = norm(b2) - norm(A2\*x2)/(norm(A2)\*norm(x2));

rel\_err(3) = norm(x3-xtrue)/norm(xtrue);

rel\_res(3) = norm(b3) - norm(A3\*x3)/(norm(A3)\*norm(x3));

rel\_err(4) = norm(x4-xtrue)/norm(xtrue);

rel\_res(4) = norm(b4) - (norm(A4\*x4)/(norm(A4)\*norm(x4)));

rel\_err(5) = norm(x5-xtrue)/norm(xtrue);

rel\_res(5) = norm(b5) - (norm(A5\*x5)/(norm(A5)\*norm(x5)));

% The relative errors are bounded by conditional number of matrix A

%inv matrix method

A1inv = inv(A1);

A2inv = inv(A2);

A3inv = inv(A3);

A4inv = inv(A4);

A5inv = inv(A5);

xi1 = A1inv\*b1;

xi2 = A2inv\*b2;

xi3 = A3inv\*b3;

xi4 = A4inv\*b4;

xi5 = A5inv\*b5;

i\_rel\_err(1) = norm(xi1-xtrue)/norm(xtrue);

i\_rel\_res(1) = norm(b1) - norm(A1\*xi1)/(norm(A1)\*norm(xi1));

i\_rel\_err(2) = norm(xi2-xtrue)/norm(xtrue);

i\_rel\_res(2) = norm(b2) - norm(A2\*xi2)/(norm(A2)\*norm(xi2));

i\_rel\_err(3) = norm(xi3-xtrue)/norm(xtrue);

i\_rel\_res(3) = norm(b3) - norm(A3\*xi3)/(norm(A3)\*norm(xi3));

i\_rel\_err(4) = norm(xi4-xtrue)/norm(xtrue);

i\_rel\_res(4) = norm(b4) - norm(A4\*xi4)/(norm(A4)\*norm(xi4));

i\_rel\_err(5) = norm(xi5-xtrue)/norm(xtrue);

i\_rel\_res(5) = norm(b5) - norm(A5\*xi5)/(norm(A5)\*norm(xi5));

%Cramer's rule

xc1 = cramer(A1,b1);

xc2 = cramer(A2,b2);

xc3 = cramer(A3,b3);

xc4 = cramer(A4,b4);

xc5 = cramer(A5,b5);

c\_rel\_err(1) = norm(xc1-xtrue)/norm(xtrue);

c\_rel\_res(1) = norm(b1) - norm(A1\*xc1)/(norm(A1)\*norm(xc1));

c\_rel\_err(2) = norm(xc2-xtrue)/norm(xtrue);

c\_rel\_res(2) = norm(b2) - norm(A2\*xc2)/(norm(A2)\*norm(xc2));

c\_rel\_err(3) = norm(xc3-xtrue)/norm(xtrue);

c\_rel\_res(3) = norm(b3) - norm(A3\*xc3)/(norm(A3)\*norm(xc3));

c\_rel\_err(4) = norm(xc4-xtrue)/norm(xtrue);

c\_rel\_res(4) = norm(b4) - norm(A4\*xc4)/(norm(A4)\*norm(xc4));

c\_rel\_err(5) = norm(xc5-xtrue)/norm(xtrue);

c\_rel\_res(5) = norm(b5) - norm(A5\*xc5)/(norm(A5)\*norm(xc5));

Explanation:

I’ve run the routine for several times, each time the resulted solution is pretty accurate. All the relative errors are residues are really small. So I would guess those methods are stable in such cases.

However, I know the inverse methods and cramer’s rule are not backward stable. I don’t understand why my implemation gives accurate result. Maybe it’s due to the normal distribution of the elements. Maybe because Matlab has updated its algorithm to a better one? Otherwise it cannot be explained.

The ill conditioned matrix should amplify the perturbation, making the relative error really large. Maybe its because the matrix A is generated using two orthogonal matrix?