Johnny Donza MEC 320

Programming Assignment 2 Matlab

Submission Number: **68857b97-a005-486d-b5c6-8f9a79dbaace**

**Task 1:**

1. The x-values without partial pivoting are displayed in the table below.

|  |
| --- |
| -1.000 |
| -12.2389 |
| -3.4328 |
| 8.1716 |
| -11.9770 |
| -3.7387 |
| 1.8041 |
| 2.6230 |
| 11.1400 |

1. The x-values with partial pivoting are displayed in the table below.

|  |
| --- |
| -1.000 |
| -12.2389 |
| -3.4328 |
| 8.1716 |
| -11.9770 |
| -3.7387 |
| 1.8041 |
| 2.6230 |
| 11.1400 |

1. There are no differences between the x values with and without partial pivoting
2. The condition number of A using a column-sum norm came out to be 182.7547, deeming matrix A as an ill-conditioned system. This means that the matrix is almost singular and that the solution can be prone to large numerical errors.

*Gauss Elimination without Partial Pivoting*

close all

clear

clc

%%Gauss Elimination without partial pivoting

%The general concepts of how to set-up the for loops for the foward elimination

%and back substitution was obtained from the pseudocode provided in the

%textbook on page 254. This part of the code was interpreted into Matlab and altered to

%better represent this particular task.

%Step 1: Create Matrix from system of equations

A = [-2 1 -9 2 2 5 1 8 -1; ...

1 -6 3 12 11 -3 -7 2 -2; ...

2 -3 -1 2 5 13 2 16 3; ...

-6 2 -3 1 2 -1 14 2 -2; ...

-1 -13 4 -4 3 7 -3 2 -3; ...

15 3 -5 7 -2 5 2 1 -1; ...

14 3 -5 7 -2 5 2 1 -1; ...

3 2 2 -5 -2 -14 6 7 -2; ...

3 2 3 -4 -2 -13 6 7 -2];

b = [6; 10; 25; -12; 18; 23; 24; 8; 9];

n = input('Number of variables =');

%Step 2: Create augmented matrix

Am = [A b];

%Step 3: Determine the size of the matrix

[nA,mA] = size(A);%nA = rows; mA = columns

[nb,mb] = size(b);%nb = rows; mb = columns

%Step 4: Determine the condition number

S = abs(sum(A)); %Finds the absolute sum of all the columns

S\_max = max(S); %Determines the maximum sum

I = inv(A); %Calculates the inverse of A

I\_sum = abs(sum(I)); %Sums the columns in I

I\_max = max(I\_sum); %Chooses the maximum sum

cond\_A = S\_max \* I\_max; %Determines the condition number

if cond\_A > 100

disp('system is ill-conditioned');

else

disp('system is well-conditioned');

end

%Step 5: Foward Elimination

for j = 1:nb %j is for columns

for i = j+1:nA %i is for rows

Am(i,:) =Am(i,:)-(Am(j,:)\* Am(i,j)/Am(j,j));

end

end

% Step 6: Back Substitution

x = zeros(n,1); %Sets up a zero matrix to fill in solution values

m = n+1; %Written to insure we call last column in augmented matrix

x(n) = Am(n,m)/Am(n,n); %Defines the value for your last variable in matrix

for i = n-1:-1:1

solution = Am(i,m); %Calls upon solution value as given in b-vector

for j = i+1:n

solution = solution - Am(i,j)\*x(j,:);

end

x(i) = solution/Am(i,i);

end

disp('[x] =');

disp(x);

%Step 7: Check to see if code works

b = A\*x;

disp('b =');

disp(b);

*Gauss Elimination with Partial Pivoting*

close all

clear

clc

%%Gauss Elimination with partial pivoting

%The general concepts of how to construct partial pivoting in Matlab was obtained from

%the pseudocode provided in the textbook on page 266. This part of the code was interpreted

%into Matlab and altered to better represent this particular task.

%Step 1: Create Matrix from system of equations

A = [-2 1 -9 2 2 5 1 8 -1; ...

1 -6 3 12 11 -3 -7 2 -2; ...

2 -3 -1 2 5 13 2 16 3; ...

-6 2 -3 1 2 -1 14 2 -2; ...

-1 -13 4 -4 3 7 -3 2 -3; ...

15 3 -5 7 -2 5 2 1 -1; ...

14 3 -5 7 -2 5 2 1 -1; ...

3 2 2 -5 -2 -14 6 7 -2; ...

3 2 3 -4 -2 -13 6 7 -2];

b = [6; 10; 25; -12; 18; 23; 24; 8; 9];

n = input('Number of variables =');

%Step 2: Create augmented matrix

Am = [A b];

%Step 3: Determine the size of the matrix

[nA,mA] = size(A);%nA = rows; mA = columns

[nb,mb] = size(b);%nb = rows; mb = columns

%Step 4: Determine the condition number

S = abs(sum(A)); %Finds the absolute sum of all the columns

S\_max = max(S); %Determines the maximum sum

I = inv(A); %Calculates the inverse of A

I\_sum = abs(sum(I)); %Sums the columns in I

I\_max = max(I\_sum); %Chooses the maximum sum

cond\_A = S\_max \* I\_max; %Determines the condition number

if cond\_A > 100

disp('system is ill-conditioned');

else

disp('system is well-conditioned');

end

%Step 5: Foward Elimination with partial pivoting

for j = 1:nb %j is for columns

%Partial Pivoting

m = n+1;

p = 1;

k = p;

pivot = Am(k,k);

for ii = k+1:1:n

pivot2 = Am(ii,k);

if pivot2 > pivot

pivot = pivot2;

p = ii;

end

end

if p ~= k

for jj = k:1:n

pivot2 = Am(p,jj);

Am(p,jj) = Am(k,jj);

Am(k,jj) = pivot2;

end

pivot2 = Am(p,m);

Am(p,m) = Am(k,m);

Am(k,m) = pivot2;

end

for i = j+1:nA %i is for rows

Am(i,:) =Am(i,:)-(Am(j,:)\* Am(i,j)/Am(j,j));

end

end

%Step 6: Back Substitution

x = zeros(n,1); %Sets up a zero matrix to fill in solution values

m = n+1; %Written to insure we call last column in augmented matrix

x(n) = Am(n,m)/Am(n,n); %Defines the value for your last variable in matrix

for i = n-1:-1:1

solution = Am(i,m); %Calls upon solution value as given in b-vector

for j = i+1:n

solution = solution - Am(i,j)\*x(j,:);

end

x(i) = solution/Am(i,i);

end

disp('[x] =');

disp(x);

%Step 7: Check to see if code works

b = A\*x;

disp('b =');

disp(b);

**Task 2:**

1. The iterative refinement portion of my code performed 8 iterations.
2. My final approximate error for the iterative refinement can be seen in the table below.

|  |
| --- |
| 2.9127 e-09 |
| 5.2728 e-09 |
| 9.5719 e-09 |
| 2.4177 e-09 |
| 2.9019 e-09 |
| 9.5106 e-09 |
| 8.3177 e-09 |
| 2.2537 e-09 |
| 7.1704 e-09 |

1. The x-values that solve the system are shown in the table below starting from x1 through x9 respectively.

|  |
| --- |
| -1.000 |
| -12.2389 |
| -3.4328 |
| 8.1716 |
| -11.9770 |
| -3.7387 |
| 1.8041 |
| 2.6230 |
| 11.1400 |

*LU-Decomposition with Iterative Refinement*

close all

clear

clc

%%LU-Decomposition with iterative refinement

%Information pertaining to the implementation of the decomposition of the A

%matrix was obtained from the pseudocode presented on page 282 in the

%textbook.

%Step 1: Create Matrix from system of equations

A = [-2 1 -9 2 2 5 1 8 -1; ...

1 -6 3 12 11 -3 -7 2 -2; ...

2 -3 -1 2 5 13 2 16 3; ...

-6 2 -3 1 2 -1 14 2 -2; ...

-1 -13 4 -4 3 7 -3 2 -3; ...

15 3 -5 7 -2 5 2 1 -1; ...

14 3 -5 7 -2 5 2 1 -1; ...

3 2 2 -5 -2 -14 6 7 -2; ...

3 2 3 -4 -2 -13 6 7 -2];

b = [6; 10; 25; -12; 18; 23; 24; 8; 9];

n = input('Number of variables =');

%Step 2: Determine the size of the matrix

[nA,mA] = size(A);%nA = rows; mA = columns

[nb,mb] = size(b);%nb = rows; mb = columns

%Step 3: Begin Decomposing A into L and U matrix

L = zeros(nA,mA);

U = zeros(nA,mA);

%Step 3.1: Foward Elimination to Determine U Matrix

for j = 1:nb %j is for columns

for i = 1:nA

U(i,j) = A(i,j);

end

end

for j = 1:nb

for i = j+1:nA %i is for row

U(i,:) =U(i,:)-(U(j,:)\* U(i,j)/U(j,j));

end

end

%Step 3.2: Determine L Matrix

for i = 1:nA

L(i,i) = 1;

end

for j = 1:nb

for i = j+1:nA

L(i,j) = A(i,j)/A(j,j);

end

end

%Step 4: Begin substitution to find solution matrix

%Step 4.1: Foward substitution with L matrix

d = zeros(n,1); %Sets up a zero matrix to fill in solution values

m = 1;

d(m,:) = b(m,:)/L(m,m);

for i = m+1:1:nA

solution = b(i,:);

for j = 1:nb

solution = solution - L(i,j)\*d(j);

end

d(i) = solution/L(i,i);

end

disp('[d\_approx] =');

disp(d);

%Step 4.2: Backwards substitution with U matrix

x\_approx = zeros(n,1); %Sets up a zero matrix to fill in approximate solution values

x\_approx(n) = d(n,:)/U(n,n); %Defines the value for your last variable in matrix

for i = n-1:-1:1

solution = d(i,:); %Calls upon solution value as given in b-vector

for j = i+1:n

solution = solution - U(i,j)\*x\_approx(j,:);

end

x\_approx(i) = solution/U(i,i);

end

disp('[x] =');

disp(x\_approx);

%Step 5: Iterative Refinement

es = zeros(n,1);

es(:,1) = .001; %units in percent

eA = zeros(n,1);

eA(:,1) = 100; %units in percent

k = 0;

while max(eA) > es(:,1)

k = k+1;

b\_approx = A \* x\_approx;

disp('b\_approx =');

disp(b\_approx);

E = b - b\_approx; %Error in b matrix

%Find the correction factor matrix x\_delta

%Foward substitution with L matrix

y = zeros(n,1); %Sets up a zero matrix to fill in solution values

m = 1;

y(m,:) = E(m,:)/L(m,m);

for i = m+1:1:nA

solution = E(i,:);

for j = 1:nb

solution = solution - L(i,j)\*y(j);

end

y(i) = solution/L(i,i);

end

disp('[y] =');

disp(y);

%Backwards substitution with U matrix

x\_delta = zeros(n,1); %Sets up a zero matrix to fill in approximate solution values

x\_delta(n) = y(n,:)/U(n,n); %Defines the value for your last variable in matrix

for i = n-1:-1:1

solution = y(i,:); %Calls upon solution value as given in b-vector

for j = i+1:n

solution = solution - U(i,j)\*x\_delta(j,:);

end

x\_delta(i) = solution/U(i,i);

end

disp('[x\_delta] =');

disp(x\_delta);

x\_approx1 = x\_delta + x\_approx;

eA = abs((x\_approx1 - x\_approx)./(x\_approx1))\*100;

x\_approx = x\_approx1;

end

x = x\_approx;

disp('x =');

disp(x);

**Task 3**

1. The x-values that solve the system are shown in the table below starting from x1 through x6 respectively.

|  |
| --- |
| 1.1409 |
| 1.0657 |
| 1.9568 |
| 1.7574 |
| -0.1836 |
| 0.2480 |

1. My code performed 9 iterations.
2. I know that this system will converge because the A matrix has been transformed into a diagonally dominant matrix by switching the rows.

*Gauss-Seidel*

close all

clear

clc

%%Gauss Seidel Method

%Step 1: Create Matrix from system of equations

A = [2 -1 -2 9 2 1;...

2 2 -2 3 11 1;...

1 1 9 -2 3 1;...

13 3 -4 -2 1 2;...

-2 -1 1 4 2 11;...

-4 -12 1 2 2 1];

b = [7; -12; 16; 13; 4; 8];

n = input('Number of variables =');

%Step 2: Define Size of the matrix

[nA,mA] = size(A);%nA = rows; mA = columns

[nb,mb] = size(b);%nb = rows; mb = columns

%Step 3: Check for convergence

for i = 1:mA

k = abs(A(i,i));

s = sum(abs(A(i,:))- abs(k)); %Checking if A-Matrix is diagnolly dominant

if s > k

disp('System may diverge');

break

end

end

%Shows that intially, the system will not converge

% Insert Diagonally Dominant Matrix

A = [13 3 -4 -2 1 2; ...

-4 -12 1 2 2 1; ...

1 1 9 -2 3 1;...

2 -1 -2 9 2 1; ...

2 2 -2 3 11 1; ...

-2 -1 1 4 2 11];

%Step 4: Define your initial guesses

x\_initial = zeros(mA,1);

x = zeros(mA,1);

%Step 5: Begin Gauss-Seidel Iteration

es = zeros(nb,1);

es(:,:) = 0.0005; %Units in percent

ea = zeros(nb,1);

ea(:,:) = 100;

k = 0;

while max(ea) > es(:,1)

k = k+1; %Calculates number of iterations

for i = 1:nA

solution = b(i,:);

for j = 1:i-1

solution = solution - A(i,j)\*x(j,:);

end

for j = i+1:mA

solution = solution - A(i,j)\*x(j,:);

end

x(i) = solution/A(i,i);

end

ea(:,1) = abs((x(:,1) - x\_initial(:,1))./(x(:,1)))\*100;

x\_initial = x;

end

disp('x =');

disp(x);

**Task 4**

1. NaN
2. 2
3. Total iterations were 17,752.
4. I was not able to find a minimum, maximum, or a saddle point. I believe the issue lies within my parabolic interpolation function but unfortunately, I did not have enough time to figure it out.
5. 0.488875 seconds.

*Steepest Ascent/Descent with Parabolic Interpolation*

tic

close all

clear

clc

%%Steepest Ascent/Descent

%Define anonymous functions

x0 = 0;

y0 = 0;

Ea = zeros(2,1);

Ea(:,:) = 100; %units in percent

I = max(Ea);

Es = .01; %units in percent

w = 0;

while I > Es

w = w+1;

f = @(x,y)10\*sin(x+3) - 6\*cos(2\*y + 2);

df\_dx = @(x,y)10\*cos(x+3);

df\_dy = @(x,y)12\*sin(2\*y + 2);

df2\_dx2 = @(x,y)-10\*sin(x + 3);

df2\_dy2 = @(x,y)24\*cos(2\*y + 2);

df2\_dxdy2 = @(x,y)10\*cos(x+3) + 12\*sin(2\*y + 2);

g = @(h) f(x0 + (df\_dx(x0,y0))\*h, y0 + (df\_dy(x0,y0))\*h);

%Initial Guesses

h0 = 0;

h1 = 0.1;

h2 = 0.2;

ea = 100;

es = .01; %units in percent

k = 0;

while ea > es

k = k+1;

a = g(h0);

I = g(h1);

c = g(h2);

h3 = (a \* (h1^2 - h2^2) + I \* (h2^2 - x0^2) + c \* (h0^2 - h1^2)) / (2\*a\*(h1-h2) + 2\*I\*(h2-h0) + 2\*c\*(h0-h1));

h\_vector(k,:) = h3;

d = g(h3);

h0 = h1;

h1 = h2;

h2 = h3;

if k > 1

ea = abs((h\_vector(k,:) - h\_vector(k-1,:))/(h\_vector(k,:)))\*100;

end

end

disp('h\_opt =');

disp (h3);

x0\_old = x0;

y0\_old = y0;

x0 = x0 + df\_dx(x0,y0)\*h3;

y0 = y0 + df\_dy(x0,y0)\*h3;

Ea1 = abs((x0 - x0\_old)/(x0))\*100;

Ea2 = abs((y0 - y0\_old)/(y0))\*100;

Ea(1,1) = Ea1;

Ea(2,1) = Ea2;

I = max(Ea);

end

disp(x0);

disp(y0);

toc