Q2a) The algorithms are printed below, in genpdiag(A,b) and geppdiag(A,b). In both functions, there is a minimum number of operations, and comparisons/swaps are done between the elements of row k and (2n+1)-k+1, where k ranges from 1 to n (and n is defined such that the matrix dimension is 2n+1)

function x = genpdiag(A,b)

% genpdiag.m Gauss elimination with no pivoting, for a matrix with only

% left and right diagonal elements.

%

% input: A is an n x n nonsingular matrix

% b is an n x 1 vector

% output: x is the solution of Ax=b.

%

%% This function assumes that the matrix is of type specified in Q2-cswk3.pdf

if(mod(length(b)-1,2) == 1)

print('This is not the right function for this matrix type!');

end

len = length(b);

n = (len-1)/2;

for k = 1:n % the diagonal entry has zeros below it due to the form of A

mult = A(len-k+1,k)/A(k,k);

% Only two entries of each row need updating since the rest are all

% zeros.

A(len-k+1,k) = A(len-k+1,k)-mult\*A(k,k);

A(len-k+1,len-k+1) = A(len-k+1,len-k+1) - mult\*A(k,len-k+1);

b(len-k+1) = b(len-k+1)- mult\*b(k);

end

x = zeros(len,1);

for k = len:-1:1

%Below the diagonal

if(k > n)

x(k) = b(k)/A(k,k);

else

x(k) = (b(k)-(A(k,len-k+1)\*x(len-k+1)))/A(k,k);

end

end

function x = geppdiag(A,b)

% geppdiag.m Gauss elimination with no pivoting, for a matrix with only

% left and right diagonal elements.

%

% input: A is an n x n nonsingular matrix

% b is an n x 1 vector

% output: x is the solution of Ax=b.

%

%% This function assumes that the matrix is of type specified in Q2-cswk3.pdf

if(mod(length(b)-1,2) == 1)

print('This is not the right function for this matrix type!');

end

len = length(b);

n = (len-1)/2;

for k = 1:n % the diagonal entry has zeros below it due to the form of A

%do a comparison and interchange

if(A(len-k+1,k) > A(k,k))

%only need to interchange two columns

A([k,len-k+1],k) = A([len-k+1,k],k);

A([k,len-k+1],len-k+1) = A([len-k+1,k],len-k+1);

b([k,len-k+1]) = b([len-k+1,k]);

end

mult = A(len-k+1,k)/A(k,k);

% Only two entries of each row need updating since the rest are all

% zeros.

A(len-k+1,k) = A(len-k+1,k)-mult\*A(k,k);

A(len-k+1,len-k+1) = A(len-k+1,len-k+1) - mult\*A(k,len-k+1);

b(len-k+1) = b(len-k+1)- mult\*b(k);

end

x = zeros(len,1);

for k = len:-1:1

%Below the diagonal

if(k > n)

x(k) = b(k)/A(k,k);

else

x(k) = (b(k)-(A(k,len-k+1)\*x(len-k+1)))/A(k,k);

end

end

GENP:

For the first for loop, it takes 7n flops and for the second loop, it takes n(3)+(n+1)\*1 flops. Total flops = 10n + 1

GEPP

This also takes 10n+1 flops, a maximum of 3n swaps and n comparisons

(b)

%% This is the tester for GENP and GEPP diagonal methods

%% First generate matrices

n = 9;

A = zeros(2\*n+1);

for i = 1:(2\*n+1)

A(i,i) = randn(1); % generate the left diagonal element

if(i ~= n+1)

A(i, 2\*n+2 - i) = randn(1);

b(i) = A(i,i) + A(i,2\*n+2-i);

else

b(i) = A(i,i);

end

end

%test

x\_np = genpdiag(A,b);

x\_pp = geppdiag(A,b);

x = A\b';

%Calculate the norms

xnp\_norm = norm(x\_np-x)/norm(x);

xpp\_norm = norm(x\_pp-x)/norm(x);

epsilon = eps('double')\*cond(A,2);

fprintf('xnp norm %ld xpp\_norm %ld epsilon %ld \n',xnp\_norm,xpp\_norm,epsilon);

The result is:

xnp norm 1.248303e-15 xpp\_norm 8.489663e-16 epsilon 1.540867e-14

Since the residue norms are low the solution converges quite well to the actual result.

(c )

This was the test code for part (c )

%% This is the tester for GENP and GEPP diagonal methods

%% First generate matrices

n = 9;

A = zeros(2\*n+1);

for i = 1:(2\*n+1)

if(i>1)

A(i,i) = randn(1); % generate the left diagonal element

else

A(1,1) = 10^-15;

end

if(i ~= n+1)

A(i, 2\*n+2 - i) = randn(1);

b(i) = A(i,i) + A(i,2\*n+2-i);

else

b(i) = A(i,i);

end

end

%test

x\_np = genpdiag(A,b);

x\_pp = geppdiag(A,b);

x = A\b';

%Calculate the norms

xnp\_norm = norm(x\_np-x)/norm(x);

npp\_norm = norm(x\_pp-x)/norm(x);

epsilon = eps('double')\*cond(A,2);

fprintf('xnp norm %ld xpp\_norm %ld epsilon %ld \n',xnp\_norm,xpp\_norm,epsilon);

This was the result

xnp norm 1.833669e-04 xpp\_norm 8.489663e-16 epsilon 5.396847e-15

To check for consistency, I retested again

xnp norm 2.565363e-02 xpp\_norm 8.489663e-16 epsilon 1.341378e-14

Here, the x\_np norm (2nd norm of difference between x and x\_np) is higher than that for GEPP by several orders of magnitude. This makes since since the A(1,1) element is very close to zero and leads to unstable GENP results. Since GEPP swaps away this element with the bigger element in the last row of the column, the instability in the solution is avoided (instability due to division by a number very close to zero)

(d)

%% This is the tester for GENP and GEPP diagonal methods

%% First generate matrices

n = 9;

A = zeros(2\*n+1);

for i = 1:(2\*n+1)

%generate d\_i's

if(i>1)

A(i,i) = randn(1); % generate the left diagonal element

else

A(1,1) = 10^-15;

end

if(i ~= n+1)

%generate a\_i's

if(i == 2\*n+1)

A(2\*n+1,1) = 10^-8;

else

A(i, 2\*n+2 - i) = randn(1);

end

%find b\_i's

b(i) = A(i,i) + A(i,2\*n+2-i);

else

b(i) = A(i,i);

end

end

%test

x\_np = genpdiag(A,b);

x\_pp = geppdiag(A,b);

x = A\b';

%Calculate the norms

xnp\_norm = norm(x\_np-x)/norm(x);

npp\_norm = norm(x\_pp-x)/norm(x);

epsilon = eps('double')\*cond(A,2);

fprintf('xnp norm %ld xpp\_norm %ld epsilon %ld \n',xnp\_norm,xpp\_norm,epsilon);

The result is:

xnp norm 1.833668e-04 xpp\_norm 8.489663e-16 epsilon 7.093788e-08

xnp norm 2.528690e-02 xpp\_norm 8.489663e-16 epsilon 4.802971e-08

Here again, the GENP method is unstable due to the presence of a small number close to zero on the diagonal (and division by that number) and also the conditional number increases by 6 orders of magnitude which means that the problem is ill conditioned due to the addition of a\_1 as 10^-8.