**ECE 7650 ASSIGNMENT REPORT**

**(Advance Matrix Algorithm)**

**Author:** Jamiu Babatunde Mojolagbe

**Department:** Electrical and Computer Engineering

**Student ID:** #7804719

**Email:** mojolagm@myumanitoba.ca

**Course:** ECE 7650

**Homework:** 3 & 4

**Sub. Date:** December 5, 2016

**QUESTION ONE (1)**

The main goal here is to implement Breadth First Search (BFS) and Reverse Cuthill-McKee (RCM) ordering algorithms. However, for purpose of compactness and code simplicity, two additional functions were created to assist in this.

The two (2) additional functions are:

1. “**csr.m**”: It implements Compressed Sparse Row format of an input matrix A. The parameters to be supplied to the function as well as the return values are shown in the function implementation shown below:

function [IA, JA] = csr(A)

%% csr.m

% Implements Compress sparse row

% of an input matrix A

%

% Parameters:

% A: A matrix

% Returns:

% IA: Row indices in CSR

% JA: Column indices

%

% Author: Jamiu Babatunde Mojolagbe

%% set parameters

[n, m] = size(A);

IA = ones(1, n+1);

%% loop through the rows and compile the CSR parameters

for i = 1:n;

nzr = nnz(A(i ,:));

IA(i+1) = nzr;

if i == 1; JA = [find(A(i ,:))];

else JA = [JA find(A(i ,:))];

end

end

IA = cumsum(IA);

end

1. “**getLowestDegree.m**”: This function gets the index of row of with the lowest degree of freedom of an input matrix A. The required parameters and return values are shown in the presented function implementation shown below:

function [r, n, m] = getLowestDegree(A)

%% getLowestDegree.m

% Get index of the row with lowest degree of freedom, i. e, neighbors

%

% Parameters:

% A: A sparse matrix in compressed-row format with each row sorted

% Returns:

% r: Index of the row with lowest degree of freedom

% n: Number of rows in A

% m: Number of columns in A

%

%% set parameters

[n, m] = size(A); % obtain the number of rows and cols of A

%% loop through the rows of A and get the non-zeros

for i = 1:n

nz(i) = nnz(A(i, :));

end

r = find(nz(:) == min(nz));

end

Implementation Note on Breadth First Search (BFS)

Breadth First Search (BFS) algorithm was implemented in the file named “**bfs.m**”. Two important issues were noticed that was not in the pseudocode and were careful fixed accordingly in the implementation.

The first issue is the choice of “i” – which is the initial search row index. The question here is “What happens if the choice of ‘i’ is beyond the maximum number of rows in A?”. To answer this question one line of code was added to ascertain that we are not going beyond the maximum number of rows.

Again, the second important issue here is the issue of ‘π’. The implemented BFS function is not only returning ‘π’ but also permutation matrix P. Hence, from proper study of the implementation of LU in MATLAB, it was observed that when built in MATLAB **lu( )** function is instructed to return the permutation matrix P, it returns P as a sparse matrix rather than a dense one. Therefore, in this implementation of BFS, a parameter is added to check if P is to be returned as a dense P or as a sparse one.

The required parameters or arguments for the function and return variable are shown in the function implementation code presented below:

function [P, pi] = bfs(A, i, retSparse)

%% bfs.m

% Implements Breadth First Search for

% adjacency graph traversal reordering permutation

%

% Parameters:

% A: A sparse matrix in compressed-row format with each row sorted

% i: An index of the first vertex (row) to start at

% retSparse: Boolean value whether to return P as sparse matrix

% Returns:

% P: A permutation matrix

% pi: A permutation list based on the ordering of the vertices traversed

%

% Author: Jamiu Babatunde Mojolagbe

%% obtain the dimension of A inorder to obtain number of rows

[n, m] = size(A);

assert(i <= n, 'Specified vertex does not exist');

pi = [i];

count = 1;

marked = zeros(n, 1);

marked(i) = 1;

S = [i];

[IA, JA] = csr(A);

while count < n

Snew = [];

for i = S

row\_start = IA(i);

row\_stop = IA(i+1);

for j = row\_start:row\_stop - 1

column = JA(j);

if marked(column) == 0

marked(column) = 1;

Snew = [Snew column];

pi = [pi column];

count = count + 1;

end

end

end

S = Snew;

end

P = zeros(n, n);

for i = 1:n

P(i, pi(i)) = 1;

end

if retSparse; P = sparse(P); end

end

Implementation Note on Reverse Cuthill-McKee (RCM)

This algorithm was implemented as a function in the file named “**rcm.m**”. As with the BFS function presented above, RCM implementation also take care of returning the permutation as ‘π’ as well as a permutation matrix P. Again, there is also choice whether to return P as a full matrix or as a sparse one. The arguments needed by this function is shown in the implementation code below:

function [P, pi] = rcm(A, retSparse)

%% rcm.m

% Implements Reverse Cuthill-McKee (RCM) Ordering of

% adjacency graph traversal reordering permutation

%

% Parameters:

% A: A sparse matrix in compressed-row format with each row sorted

% retSparse: Boolean value whether to return P as sparse matrix

% Returns:

% P: A permutation matrix

% pi: A permutation list based on the ordering of the vertices traversed

%

% Author: Jamiu Babatunde Mojolagbe

%% obtain the dimension of A inorder to obtain number of rows

[r, n, m] = getLowestDegree(A);

i = min(r);

pi = [i];

count = 1;

marked = zeros(n, 1);

marked(i) = 1;

S = [i];

[IA, JA] = csr(A);

while count < n

Snew = [];

for i = S

row\_start = IA(i);

row\_stop = IA(i+1);

% loop over the adjacency nodes

for j = row\_start:row\_stop - 1

column = JA(j);

if marked(column) == 0

marked(column) = 1;

Snew = [Snew column];

count = count + 1;

end

end

end

pi = [pi Snew];

S = Snew;

end

pi = fliplr(pi);

P = zeros(n, n);

for i = 1:n

P(i, pi(i)) = 1;

end

if retSparse; P = sparse(P); end

end

**QUESTION TWO (2)**

The implementation for this question are divided into three parts for the sake of convenience and code readability. However, according to the question, the task here is to test the efficiency and/or time taken to solve an input matrix A, when the matrix A is:

1. Original unpermitted matrix A
2. Bread First Search (BFS) symmetric permutation is applied to A
3. Reverse Cuthill-McKee symmetric permutation is applied to A

**Part 1**: This part test the implementation of Bread First Search (BFS), Reverse Cuthill-McKee (RCM) ordering algorithms and MATLAB built-in routine for performing permutation on an input matrix A during LU factorization. This parts test