Table of Contents

Problem 1

Implement Gaussian elimination with partial pivoting to solve

```
%   (2)x1 + (0)x2 +  (1)x3 +  (-1)x4 = 6
%   (6)x1 + (3)x2 +  (2)x3 +  (-1)x4 = 15
%   (4)x1 + (3)x2 + (-2)x3 +   (3)x4 = 3
%  (-2)x1 + (6)x2 +  (2)x3 + (-14)x4 = 12
clc
clear
% Define coefficient matrix based on problem. Has to be square.
A = [ 2, 0,  1,  -1;
      6, 3,  2,  -1;
      4, 3, -2,   3;
     -2, 6,  2, -14];

% Define b vector based on problem. Must have same number of rows as
 A.
b = [ 6;
     15;
      3;
     12];

% Created augmented matrix AM by appending A and b.
AM = [A b];

[numRows, numCols] = size(A);

% Start partial pivoting
for i = 1: numCols - 1
    % Find max element value and index in column under pivot element
    [maxEl, indexMaxEl] = max(abs(AM(i+1:end,i)));

    % The first element of this subvector corresponds to the ith
+1st,ith
    % element of AM so the corresponding row index in AM will be
    % row i + indexMaxEl
    indexMaxEl = i + indexMaxEl;

    pivot = AM(i,i);
    % Compare pivot element and max element. Swap if necessary
    if abs(maxEl) > abs(pivot)
        % Create temp variable holding values in row with max element
        temp = AM(indexMaxEl, :);
        % Swap rows
```

```
        AM(indexMaxEl,:) = AM(i,:);
        AM(i,:) = temp;
    end
    % Find mulipliers to zeros under pivots
    multipliers = -AM(i+1:end,i)/AM(i,i);
    % Create lower triangular matrix to zero under pivots
    Li = eye(numRows);
    Li(i+1:end,i) = multipliers;
    % multiply Li*AM to zero under pivot and update AM for next pass
    AM = Li*AM;
end %for i = 1: numCols - 1
% End partial pivoting

% Solution using partial pivoting
solVecPP = BackSub(AM, numRows)

% Solution using MATLAB's backslash operator
matlabSol1 = A\b

disp(matlabSol1-solVecPP)
```

Problem 2

```
%Implement Gaussian elimination with scaled partial pivoting to solve

%       (pi)x1 + (sqrt(2))x2 +       (-1)x3 +        (1)x4 = 0
%   (exp(1))x1 +      (-1)x2 +        (1)x3 +        (2)x4 = 1
%        (1)x1 +       (1)x2 + (-sqrt(3))x3 +        (1)x4 = 2
%       (-1)x1 +      (-1)x2 +        (1)x3 + (-sqrt(5))x4 = 3

% Define coefficient matrix based on problem. Has to be square.
A = [    pi, sqrt(2),       -1,       1;
     exp(1),      -1,        1,       2;
          1,       1, -sqrt(3),       1;
         -1,      -1,        1, -sqrt(5)];

% Define b vector based on problem. Must have same number of rows as
 A.
b = [0:3]';

% Created augmented matrix AM by appending A and b.
AM = [A b];

[numRows, numCols] = size(A);

% Build vector containing largest elements from each row of A
for i = 1 : numRows
    s(i) = max(abs(A(i,:)));
end
s = s'; % Transpose for later use

% Start scaled partial pivoting
for i = 1: numCols - 1
```

```
    % Scale elements in desired colum for comparison
    testVec = AM(i:end,i)./s(i:end,1);

    % Find row with the largest element
    [maxEl, maxElIndex] = max(abs(testVec));

    % The first element of testVec corresponds to the ith,ith element
 in
    % AM, therefore, the row index in AM that corresponds to the max
    % element will be row i + maxElIndex - 1.
    maxElIndex = i + maxElIndex - 1;

    pivot = testVec(1);

    % Compare scaled pivot element and max element. Swap if necessary
    if abs(maxEl) > abs(pivot)
        % Create temp variable holding values in row with max element
        temp = AM(maxElIndex, :);
        % Swap rows
        AM(maxElIndex,:) = AM(i,:);
        AM(i,:) = temp;
    end
    AM; % Remove ;  to see if row swap happened
    % Find mulipliers to zeros under pivots
    multipliers = -AM(i+1:end,i)/AM(i,i);
    % Create lower triangular matrix to zero under pivots
    Li = eye(numRows);
    Li(i+1:end,i) = multipliers;
    % multiply Li*AM to zero under pivot and update AM for next pass
    AM = Li*AM;
end
% Solution using scaled partial pivoting
solVecSP = BackSub(AM, numRows)

% Solution from matlab's backslash operator
matlabSol2 = A\b

disp(matlabSol2-solVecSP)
```

Backsubstitution Function

```
function x = BackSub(AM, numRows)
    U = AM(:,1:end-1);
    b = AM(:,end);
    x = zeros(numRows,1);
    x(numRows) = b(numRows)/U(numRows,numRows);
    for i = 1: numRows-1
        AM;
        j= numRows - i;
        UsubT = U(j,j:end)';
        xsub = x(j+1:end,:);
        bsub = b(j);
        x(numRows-i) = (bsub-dot(UsubT(2:end,:),xsub))./UsubT(1);
```

```
    end
    x;
end
```

```
solVecPP =

    2.0000
    0.0000
    1.0000
   -1.0000


matlabSol1 =

    2.0000
   -0.0000
    1.0000
   -1.0000

   1.0e-14 *

    0.1110
   -0.0761
   -0.1776
   -0.0666
```

Published with MATLAB® R2018b