MATLAB assignment 7

Introduction to Linear Algebra (Week 7) **Fall, 2019**

1. (*Programming in* MATLAB (*continued*))

   Before starting, it is highly recommended that look around the materials of the last week.

   A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follows the conditional statement, or to skip these commands. A very simple form of conditional statement, `if-else` statement, is as follows:

   ```
   if    conditional_expression                    (CE)
         ⟨a group 1 of MATLAB commands⟩            (CMD1)
   else
         ⟨a group 2 of MATLAB commands⟩            (CMD2)
   end
   ```

   MATLAB read the above code line-by-line from `if` to the `end`. If (CE) is **True**, then (CMD1) is executed but if (CE) is **False** then MATLAB skips (CMD1) and executes (CMD2). Furthermore, if there are nothing to execute when (CE) is **False** then `pass` can be used for the place (CMD2), or you can just skip the `else` statements. Sometimes it can be written as a nested `if-else if-else` form:

   ```
   if    conditional_expression 1                  (CE1)
         ⟨a group 1 of MATLAB commands⟩            (CMD1)
   else
         if    conditional_expression 2            (CE2)
               ⟨a group 2 of MATLAB commands⟩      (CMD2)
         else
               ⟨a group 3 of MATLAB commands⟩      (CMD3)
         end
   end
   ```

   Similar to the simple case, MATLAB executes (CMD1) if (CE1) is **True**. But in the next steps, it is more complicated. If the (CE1) and (CE2) are **False** and **True**, respectively, then the (CMD2) is executed and (CMD1) is skipped. If both of (CE1) and (CE2) are **False**, then only (CMD3) is executed.

This is an example of an `if-else if-else` statement:

```
if n > 0
    disp('The n is positive number');
else
    if n = 0
        disp('The n is 0');
    else
        disp('The n is negative number');
    end
end
```

Problems.

(a) Considering the items below, write a function file to find the reduced row echelon form of an $m \times n$ matrix A of $\text{rank}(A) = m$ such that A can be reduced to row echelon form by Gaussian elimination **without row interchanges**. Check your result by applying this function for the augmented matrix given in the Example 6 of the Section 2.1 of the textbook.

- Make a new function file with a function name `rerowef`. Make `A` an input to the function, and the reduced row echelon form `rref_A` of `A` the output.
- Use nested loops with the `if` statement as many times as necessary to perform the forward and backward phase without using row interchanges.
- Note that when you use the command `break` together with a `if` statement in a loop(`for` or `while` loop), it terminates the execution of the loop.

solution.

```
function [rref_A] = rerowef(A)
%- Finding Reduced Row Echelon Form without Row Interchanges -%
[nrow, ncol] = size(A); % The number of rows and columns of the matrix A.
rref_A = A;             % In order to initialize it, copy A to rref_A.

%---- Forward Phase ----%
for i=1:nrow
    % Find the first nonzero entry of the ith row.
    for k=i:ncol
        if rref_A(i,k) ~= 0
            break % Terminates the execution of the loop.
        end
    end
    rref_A(i,:) = (1/rref_A(i,k)) * rref_A(i,:);
    % Normalize the pivot (i,k)-entry by 1 to the ith row.
    if i ~= nrow
        for j=(i+1):nrow
            rref_A(j,:) = ((-rref_A(j,k)) * rref_A(i,:)) + rref_A(j,:);
            % Add minus (j,k)-entry times the ith row to the jth row.
        end
    end
end

%---- Backward Phase ----%
for i=nrow:-1:2
    % Find the first nonzero entry of the ith row.
    for k=i:ncol
        if rref_A(i,k) ~= 0
            break % Terminates the execution of the loop.
        end
    end
    for j=(i-1):-1:1
        rref_A(j,:) = ((-rref_A(j,k)) * rref_A(i,:)) + rref_A(j,:);
        % Add minus (j,k)-entry times the ith row to the jth row.
    end
end
```

(b) Considering the items below, write a function file to find the LU-decomposition of an invertible $n \times n$ matrix A such that A can be reduced to row echelon form by Gaussian elimination **without row interchanges**. Check your result by applying this function for the matrix given in the Example 2 of the Section 3.7.

- Make a new function file with a function name `ludecomp`. Make `A` an input to the function, and `L` and `U` the outputs.

solution.

```
function [L, U] = ludecomp(A)
% Find the LU-decomposition without using row interchanges.

%- Finding the LU-decomposition without Row Interchanges -%
[nrow, ncol] = size(A); % The number of rows and columns of the
    matrix A.
U = A; L = eye(ncol); % Initialization of U and L.

%---- Forward Phase ----%
for i=1:nrow

    % Find the first nonzero entry of the ith row.
    for k=i:ncol
        if U(i,k) ~= 0
            break % Terminates the execution of the loop.
        end
    end

    temp1 = U(i,k); % Save U(i,k) in temp.
    U(i,:) = (1/temp1) * U(i,:);
    % Normalize the pivot (i,k)-entry by 1 to the ith row.
    L(i,i) = (1/temp1)^(-1);
    % Place the reciprocal of the multiplier in that position in U.

    if i ~= nrow
        for j=(i+1):nrow
            temp2 = U(j,k); % Save U(j,k) in temp2.
            U(j,:) = ((-temp2) * U(i,:)) + U(j,:);
            % Add minus (j,k)-entry times the ith row to the jth
                row
            L(j,i) = -(-temp2);
            % Place the negative of the multiplier in that position
                 in U.
        end
    end
end
```