Numerical Mathematics: Homework 2

Problem 1

Recall (from your linear algebra class) the singular value decomposition of a matrix $A \in \mathbb{R}^{n\times n}$:

$$A = U\Sigma V^T$$

where $U, V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, and

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}, \quad \sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n > 0.$$

is a diagonal matrix. Find an expression of the condition number $\|A\| \left\|A^{-1}\right\|$ in terms of the singular values. Also, what is the condition number of a unitary matrix? What does this mean from a computational standpoint?

Solution. We first find an expression for $\|A\|$. Observe:

$$\begin{aligned} \|A\|^2 &= \max_{\|x\|=1} \|Ax\|^2 \\ &= \max_{\|x\|=1} x^T A^T Ax \\ &= \max_{\|x\|=1} x^T V\Sigma^T U^T U\Sigma V^T x \\ &= \max_{\|x\|=1} y^T \Sigma^2 y, \quad y = V^T x \\ &= \max_{\|x\|=1} \sigma_1^2 y_1^2 + \ldots + \sigma_n^2 y_n^2 \end{aligned}$$

Let us study this formula thorougly. How should we choose y so that we maximize the right hand side? Well $y_1^2 + \ldots + y_n^2 = 1$, so we better put all our energy on the first variable, i.e. set $y_1 = 1$ and the remaining entries equal to zero. This gives $\|A\|^2 = \sigma_1^2$ or $\|A\| = \sigma_1$. Doing the same process for the inverse gives $\left\|A^{-1}\right\| = 1/\sigma_n$. Hence,

$$\|A\| \left\|A^{-1}\right\| = \frac{\sigma_1}{\sigma_n}$$

The condition number of an unitary matrix is equal to unity. □

Problem 2

Make a function to solve a lower triangular system using forward substitution problem. Call this function `solve_lowertriangular.m`. It should take in two arguments: the first argument is a square matrix A and the second argument a vector b. The output should be x. Do the same for an upper triangular system. Call this function `solve_uppertriangular.m`.

Obviously, do not use the backslash operator (only can be used to checking purposes). It is important to initialize an array of zeros first. Matab allows one to dynamically extend the length of a vector dynamically, but this slows down the code.
Can you give a rough estimate of what the computational complexity is of this algorithm depending on problem size? Use big-O notation.

Solution. Shown below is the code.

```
function [ x ] = solve_lowertriangular( A, b)
% - Numerical mathematics course 2019 -

% Find length of vector
n = length(b);

% Initialize a vector of size n.
x = zeros(n,1);

% The forward substitution
x(1) = b(1)/A(1,1);
for k=2:n
    % Instead of writing an inner for loop, we can use Matlab's convenient
    % syntax to extract a sub-part of a matrix as follows:
    x(k) = (b(k) - A(k,1:k-1)*x(1:k-1))/ A(k,k);
end


end
```

```
function [ x ] = solve_uppertriangular( A, b)
% - Numerical mathematics course 2019 -

% We can use our solver for a lower triangular system by using an exchange
% permutation.
x = solve_lowertriangular(A(end:-1:1,end:-1:1),b(end:-1:1));

% Reverse the order of the entries again.
x = x(end:-1:1);


end
```

□

Problem 3

Make a function that finds the LU factorization of a matrix A. Call this function `find_LU.m`. It should take as argument a matrix A and return two outputs: the first one should be L and the second one is U. If the algorithm encounters a zero pivot, make the function print a message. Obviously you should break the loop, because it make no sense to continue after that.

Solution. Shown below is the code.

```
function [L, U] = find_LU(A)
% - Numerical mathematics course 2019 -



% Initialize an array for L, the A will be converted to U.
L = eye(size(A));


for k = 1:(size(A,1)-1)

    % update the entries in L:
    if A(k,k) ≠ 0
```

```
        L(k+1:end,k) = A(k+1:end,k) / A(k,k);
    else
        disp('Encountered a zero pivot. LU factorization is not possible. Ignore the results.')
        break;
    end

    % update the entries of A (or our future U):
    A(k+1:end,k+1:end) = A((k+1):end,(k+1):end) -  L(k+1:end,k) * A(k,(k+1):end);
    A(k+1:end,k) = 0;

end

U = A;




end
```

□

Problem 4

Use the code written in problem 1 and problem 2 to close the deal. Write a function called `solve_linearsystem.m` that solves a square linear system. It should take in two arguments: the first argument is a square matrix A and the second argument a vector b. The output should be x. You must call the functions you have made in problems 1 and 2 in your code.

Solution. Shown below is the code.

```
function [ x ] = solve_linearsystem( A, b )
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here

% Step 1: Find the LU factorizationn
[L,U] = find_LU(A);

% Step 2: Forward substitution of L
y = solve_lowertriangular(L,b);

% Step 3: Back substitution of U
x = solve_uppertriangular(U,y);

end
```

□

(extra) Problem 5

Repeat problems 3 and 4, but now add partial pivoting to it. Call the new functions `find_LUpartialpivot.m` and `solve_linearsystems_partialpivot.m` respectively. Note that there is NO NEED to explicitly construct a permutation matrix!

Solution. Shown below is the code.

```
function [ P,L,U ] = find_LUpartialpivot( A)
```

```
%FIND_LUPARTIALPIVOT Summary of this function goes here
%   Detailed explanation goes here

% initialize permutatio matrix, start with Identity
P = 1:size(A,1);

% Initialize an array for L, the A will be converted to U.
L = eye(size(A));


for k = 1:(size(A,1)-1)


    %%%  Perform an exchange permutation (i.e. the pivoting part) %%%
    % Find the index with the largest entry
    [¬, l] = max(abs(A(k:end,k))); l = l+k-1;

    % update permutation matrix
    P([l k]) = P([k l]);
    % exchange rows of A
    A([k l],:) = A([l k],:);

    % exchange rows of L (notice that in the first iterate it just exhanges zeros )
    L([k l],:) =  L([l k],:);  L(:,[k l]) =  L(:,[l k]);


    %%% Perform the gaussian elimination step %%%
    % update the entries in L:
    L(k+1:end,k) = A(k+1:end,k) / A(k,k);

    % update the entries of A (or our future U):
    A(k+1:end,k+1:end) = A((k+1):end,(k+1):end) -  L(k+1:end,k) * A(k,(k+1):end);
    A(k+1:end,k) = 0;

end

U = A;




end
```

```
function [ x ] = solve_linearsystems_partialpivot( A,b )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here


% Step 1: Find the LU factorizationn
[P,L,U] = find_LUpartialpivot(A);


% Step 2: Forward substitution of L
y = solve_lowertriangular(L,b(P));

% Step 3: Back substitution of U
x = solve_uppertriangular(U,y);

end
```

□