1 Xue Problems

1. (a) Begin with

$$\left[\begin{array}{cccc|c} 2 & 1 & 1 & 4 & 4 \\ -3 & 1 & 3 & 2 & 3 \\ -5 & -1 & 2 & 5 & 8 \\ 4 & 2 & 3 & 1 & 1 \end{array}\right] \mapsto \left[\begin{array}{cccc|c} 2 & 1 & 1 & 4 & 4 \\ 0 & 0.5 & 4.5 & 8 & 9 \\ 0 & 1.5 & 4.5 & 15 & 18 \\ 0 & 0 & 1 & -7 & -7 \end{array}\right] \mapsto \left[\begin{array}{cccc|c} 2 & 0 & -8 & -12 & -14 \\ 0 & 0.5 & 4.5 & 8 & 9 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & -7 & -7 \end{array}\right]$$

$$\mapsto \left[\begin{array}{cccc|c} 2 & 0 & 0 & -4 & -6 \\ 0 & 0.5 & 0 & -3.5 & -4.5 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right] \mapsto \left[\begin{array}{cccc|c} 2 & 0 & 0 & 0 & -2 \\ 0 & 0.5 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

And therefore $x = [-1, 2, 0, 1]^T$.

(b) Now using LU factorization,

$$\left[\begin{array}{cccc} 2 & 1 & 1 & 4 \\ -3 & 1 & 3 & 2 \\ -5 & -1 & 2 & 5 \\ 4 & 2 & 3 & 1 \end{array}\right] \mapsto \left[\begin{array}{cccc} 2 & 1 & 1 & 4 \\ 0 & 0.5 & 4.5 & 8 \\ 0 & 1.5 & 4.5 & 1 \\ 0 & 0 & 1 & -7 \end{array}\right] \mapsto \left[\begin{array}{cccc} 2 & 1 & 1 & 4 \\ 0 & 0.5 & 4.5 & 8 \\ 0 & 0 & -9 & -9 \\ 0 & 0 & 1 & -7 \end{array}\right] \mapsto \left[\begin{array}{cccc} 2 & 1 & 1 & 4 \\ 0 & 0.5 & 4.5 & 8 \\ 0 & 0 & -9 & -9 \\ 0 & 0 & 0 & -8 \end{array}\right] = U$$

And

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1.5 & 1 & 0 & 0 \\ -2.5 & 3 & 1 & 0 \\ 2 & 0 & -1/9 & 1 \end{bmatrix}$$

Back solving for $Ly = b$ gives $y = [4, 9, -9, -8]^T$ and consequently $Ux = y$ yields $x = [-1, 2, 0, 1]^T$ which is consistent with part (a).

2. We will accomplish this is no more than $4 - 1$ steps.

- Step 1: After pivoting and elimination of column 1, we obtain

$$A^{(1)} = \begin{bmatrix} -5 & -1 & 2 & 5 \\ 0 & -.4 & 1.8 & -1 \\ 0 & .6 & 1.8 & 6 \\ 0 & 1.2 & 4.6 & 5 \end{bmatrix}, L_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -.6 & 1 & 0 & 0 \\ 0.4 & 0 & 1 & 0 \\ 0.8 & 0 & 0 & 1 \end{bmatrix}, P_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Step 2: Here, we swap row 2 and 4

$$A^{(2)} = \begin{bmatrix} -5 & -1 & 2 & 5 \\ 0 & 1.2 & 4.6 & 5 \\ 0 & 0 & -.5 & 3.5 \\ 0 & 0 & 3.33 & 0.67 \end{bmatrix}, L_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -.5 & 1 & 0 \\ 0 & 0.33 & 0 & 1 \end{bmatrix}, P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

- Step 3: Finally, after swapping rows 3 and 4,

$$U = A^{(3)} = \begin{bmatrix} -5 & -1 & 2 & 5 \\ 0 & 1.2 & 4.6 & 5 \\ 0 & 0 & 3.33 & 0.67 \\ 0 & 0 & 0 & 3.6 \end{bmatrix}, L_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & .15 & 1 \end{bmatrix}, P_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

After computing $\tilde{L}_3 = L_3, \tilde{L}_2 = P_3 L_2 P_3$, and $\tilde{L}_1 = P_3 P_2 L_1 P_2 P_3$, we have

$$\tilde{L}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & .15 & 1 \end{bmatrix}, \tilde{L}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & .33 & 1 & 0 \\ 0 & -.5 & 0 & 1 \end{bmatrix}, \tilde{L}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.8 & 1 & 0 & 0 \\ -.6 & 0 & 1 & 0 \\ .4 & 0 & 0 & 1 \end{bmatrix},$$

we get that

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -.8 & 1 & 0 & 0 \\ .6 & -.33 & 1 & 0 \\ -.4 & .5 & -.15 & 1 \end{bmatrix}, P = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

And indeed, $PA = LU$.

3. Testing questions 1 and 2 using my code is simple. Simply running x = matSolve(A,b,'pivot') and x = matSolve(A,b) will give the desired results. The output of my Vandermonde matrix code is as follows

```
For n = 9
 Error without pivoting = 1.444896e-11
 with values as large as 5.318478e+05, 3.936398e+03 in L,U respectively
 Error with pivoting = 6.449528e-17
   with values as large as 1.000000e+00, 2.000000e+00 in L,U respectively

For n = 19
 Error without pivoting = 4.850979e-02
 with values as large as 1.379967e+15, 6.689227e+08 in L,U respectively
 Error with pivoting = 8.174859e-17
   with values as large as 1.000000e+00, 2.000000e+00 in L,U respectively

For n = 29
 Error without pivoting = 1.365963e+08
 with values as large as 5.641403e+25, 5.928481e+14 in L,U respectively
 Error with pivoting = 8.894552e-17
   with values as large as 1.000000e+00, 2.000000e+00 in L,U respectively

For n = 39
 Error without pivoting = 8.641654e+18
 with values as large as 6.148631e+35, 3.396194e+22 in L,U respectively
 Error with pivoting = 1.079193e-16
   with values as large as 1.000000e+00, 2.000000e+00 in L,U respectively
```

As easily seen by the errors, it is possible for LU factorization without pivoting to be numerically unstable (indeed this is the case for partial pivoting as well, but it is not evident here). In particular, as n gets larger, this problem becomes more unstable. In this problem, the instability comes from the fact that the Vandermonde matrix has many values very small in modulus that are then amplified in subsequent steps.

4. (a) Adding in the command listed in the homework, my code output the following

```
For n = 9,the matrix is diagonally dominant
Error without pivoting = 1.073526e-16
 with values as large as 1.000000e+00, 1.420441e+01 in L,U respectively
 Error with pivoting = 1.073526e-16
   with values as large as 1.000000e+00, 1.420441e+01 in L,U respectively

For n = 19,the matrix is diagonally dominant
Error without pivoting = 1.466103e-16
 with values as large as 1.000000e+00, 2.451915e+01 in L,U respectively
 Error with pivoting = 1.466103e-16
   with values as large as 1.000000e+00, 2.451915e+01 in L,U respectively

For n = 29,the matrix is diagonally dominant
Error without pivoting = 3.660111e-16
 with values as large as 1.000000e+00, 3.475051e+01 in L,U respectively
 Error with pivoting = 3.660111e-16
   with values as large as 1.000000e+00, 3.475051e+01 in L,U respectively

For n = 39,the matrix is diagonally dominant
Error without pivoting = 2.865298e-16
 with values as large as 1.000000e+00, 4.493971e+01 in L,U respectively
 Error with pivoting = 2.865298e-16
   with values as large as 1.000000e+00, 4.493971e+01 in L,U respectively
```

I ran a check to verify each matrix was indeed (column) diagonally dominant. This means that pivoting will not be required. This is verified by the fact that the errors in LU factorization without pivoting appear to be around machine epsilon (and identical to those using pivoting).

(b) As requested, my output is

```
The norm of the (absolute) error between mine and Matlab's solution is 6.621178e+01
The error of ||PA-LU||/||A|| is 1.282432e+12
The error of ||b-Ax||/||b|| is 9.456378e-01
```

It is quite clear that my solution is not accurate. It is not close to Matlab's solution, nor does it provide a good relative forward error either.

(c) Using LU factorization with complete pivoting, the provide script gave me matrices satisfying $PAQ = LDU$ where P and Q are permutation matrices, L and U are unit

lower and upper triangular respectively, and D is diagonal. To use these to solve $Ax = b$, we simply perform the following:

i. Solve $Lv = Pb$ which is just forward substitution (and a permutation)
ii. Solve $Dw = v$ which runs in $\mathcal{O}$ since D is diagonal
iii. Solve $Uy = w$ which is just backwards substitution
iv. Compute $Qy = x$ which is simply another permutation.

After these 4 steps, I obtained the following errors

```
The error of ||PA-LU||/||A|| is 0.000000e+00
The error of ||b-Ax||/||b|| is 0.000000e+00
```

which are pretty good.

(d) There are some, albeit not that many, matrices that cause instability in the LU factorization with partial pivoting algorithm. It is in these cases that complete pivoting must be used. However, because complete pivoting is of a higher complexity class, completely changes memory access, and is only really necessary for matrices designed to mess up partial pivoting, it is not used in practice.

5. (a) We assume that A is a matrix such that all entries below its $k - th$ subdiagonal are zero with everything else being nonzero. To compute its arithmetic cost for LU factorization, we will compute only the cost of elimination that differs from regular LU factorization. Any subsequent steps, such as solving $Ly = b, Ux = y$ is precisely the same as before. Note before-hand that after the kth column, the remaining columns have no zeros. Thus, the only portion of improvement is on the elimination of the first k columns. The arithmetic cost of eliminating the first k columns is exactly $(k-1)[\sum_{i=1}^{k} 2(n-i+1)$ since each column will require the zero'ing of $(k-1)$ rows and elimination of a row i requires $2(n-i+1)$ operations. This sum can be reduced to $k[2(k-1)(n+1) - (k+1)]$ which is marginally better than LU factorization-elimination depending on k.

(b) Now assume that A is a banded matrix. That is, in addition to the assumptions before, all entries above the kth superdiagonal are also zero. In this case, the elimination of a particular row is made simpler. Indeed, now the elimination of a row is made to be a constant $2(n-k)$ which gives a total of $2(k-1)(n-k)k$ arithmetic operations for the first k rows (steps).
However, it should be noted that unless A is a diagonal matrix, the complexity of LU factorization remains the same.

6. (a) Let A be a SPD $n \times n$ matrix and $x = e_k$ where $1 \leq k \leq n$. Then the argument

$$a_{kk} = e_k^T A e_k = x^T A x \geq 0$$

shows that $a_{kk} \geq 0$ for all k. That is, the diagonal elements of symmetric positive definite matrix must be positive.

(b) The calculation of l_{jj} for the Cholesky factorization is as follows

$$l_{jj} = \left(a_{jj} = \sum_{k=1}^{j-1} l_{jk}^2 \right)^{1/2}$$

We can rearrange this to be

$$\sum_{k=1}^{j} l_{jk}^2 = a_{jj}$$

for all j, k. Therefore,

$$l_{jk} \leq \max_j \sqrt{a_{jj}}$$

So the coefficients of L are uniformly bounded.

(c) The arithmetic cost of Cholesky factorization can be broken into two parts: the diagonal and the off-diagonal elements. Each diagonal clearly takes $2(j-1)+1$ operations, while off diagonal elements require $[2(j-1)+1](n-j)$ operations for a row of L. This sum can be simplified as follows

$$\begin{aligned}
\sum_{j=1}^{n} 2(j-1) + 1 + [2(j-1)+1](n-j) &= \sum_{j=1}^{n} (2j-1)(1+n-j) \\
&= 2\sum_{j=1}^{n} j(n+1-j) - \sum_{j=1}^{n} (n+1-j) \\
&= \frac{(2n+1)(n)(n+1)}{6} + \frac{n(n+1)}{2} - \frac{n(n+1)}{2} \\
&= \frac{(2n+1)(n)(n+1)}{6}
\end{aligned}$$

After some tedious algebra (indeed, I skipped a few steps here for brevity). For fun, a combinatorial argument would work here as well. Note that the computed arithmetic cost is equal to the sum of squares from 1 to n. Counting the operations on a row by row basis would yield the same results. Alas, I digress. One proof should be just fine.

(d) I am not sure why we were asked this question, other than to implement Cholesky factorization once. Nonetheless, my L factor was

```
(1,1)       2.0000
(2,1)      -0.5000
(4,1)      -0.5000
(2,2)       1.9365
(3,2)      -0.5164
(4,2)      -0.1291
(5,2)      -0.5164
(3,3)       1.9322
(4,3)      -0.0345
```

```
(5,3)      -0.1380
(6,3)      -0.5175
(4,4)       1.9319
(5,4)      -0.5546
(6,4)      -0.0092
(7,4)      -0.5176
(5,5)       1.8457
(6,5)      -0.5833
(7,5)      -0.1555
(8,5)      -0.5418
(6,6)       1.8417
(7,6)      -0.0519
(8,6)      -0.1716
(9,6)      -0.5430
(7,7)       1.9249
(8,7)      -0.5679
(9,7)      -0.0146
(8,8)       1.8315
(9,8)      -0.6014
(9,9)       1.8285
```

2 Appendix

2.1 Script files

2.1.1 Question 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 3
%
% Question
%   Problem 3
%
% Function Dependencies
%   pluFact.m
%   pluExtra.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

n = [9, 19, 29, 39];
for j = 1:length(n)
    u = -cos((0:n(j))/n(j)*pi);
    A = vander(u);
    matSize = length(A);

    [A1] = pluFact(A); %(call your LU factorization without pivoting)
    [A2,p] = pluFact(A,'pivot'); %(call your LU factorization with partial pivoting)

    %Adjust my outputs to match expected outputs
    [~,L1,U1] = pluExtract(A1,1:matSize);
    [P,L2,U2] = pluExtract(A2,p);

    %Compute error norms
    normReg = norm(A-L1*U1)/norm(A);
    normPivot = norm(P*A-L2*U2)/norm(A);
    nonZReg = [max(abs(nonzeros(L1))) max(abs(nonzeros(U1)))];
    nonZPivot = [max(abs(nonzeros(L2))) max(abs(nonzeros(U2)))];
```

```
    fprintf('For n = %d\n\t Error without pivoting = %e\n\t\t with values as large as %e, %e i
    fprintf('\t Error with pivoting = %e\n \t\t with values as large as %e, %e in L,U respecti
end
```

2.1.2 Question 4

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 3
%
% Question
%   Problem 4
%
% Function Dependencies
%   pluFact.m
%   pluExtra.m
%   matSolve.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

%% Part A
n = [9, 19, 29, 39];
for j = 1:length(n)
    u = -cos((0:n(j))/n(j)*pi);
    A = vander(u);
    A = A + diag((5:5+n(j))');
    matSize = length(A);

    diagA = diag(abs(A));
    colSum = sum(tril(abs(A),-1),2) + sum(triu(abs(A),1),2);
    diagDom = 1;
    for i = 1:matSize
        if diagA(i) < colSum
            diagDom = 0;
        end
    end

    [A1] = pluFact(A); %(call your LU factorization without pivoting)
    [A2,p] = pluFact(A,'pivot'); %(call your LU factorization with partial pivoting)

    %Adjust my outputs to match expected outputs
```

```
    [~,L1,U1] = pluExtract(A1,1:matSize);
    [P,L2,U2] = pluExtract(A2,p);

    %Compute error norms
    normReg = norm(A-L1*U1)/norm(A);
    normPivot = norm(P*A-L2*U2)/norm(A);
    nonZReg = [max(abs(nonzeros(L1))) max(abs(nonzeros(U1)))];
    nonZPivot = [max(abs(nonzeros(L2))) max(abs(nonzeros(U2)))];

    fprintf('For n = %d,',n(j))
    if diagDom == 1
        fprintf('the matrix is diagonally dominant\n\t')
    else
        fprintf('\n\t')
    end
    fprintf('Error without pivoting = %e\n\t\t with values as large as %e, %e in L,U respective
    fprintf('\t Error with pivoting = %e\n \t\t with values as large as %e, %e in L,U respecti
end

%% Part b
clear
clc
close all;

n = 100;
A = eye(n);
A(:,n) = ones(n,1);
for i = 2:n
    A(i,1:i-1) = ones(1,i-1)*-1;
end
b = A*ones(n,1);

matSolu = A\b;
mySolu = matSolve(A,b);

[infoMat,p] = pluFact(A,b);
[P,L,U] = pluExtract(infoMat,p);

fprintf('The norm of the (absolute) error between my solution and Matlab''s solution is %e\n',n
myError1 = norm(P*A-L*U)/norm(A);
myError2 = norm(b-A*mySolu)/norm(b);
fprintf('The error of ||PA-LU||/||A|| is %e\nThe error of ||b-Ax||/||b|| is %e\n',myError1,myEr

%% Part C
[P,Q,L,D,U] = HW3_lucp(A);
```

```
b = P*b;
v = L\b;
w = D\v;
y = U\w;
x = Q*y;

myError1 = norm(P*A*Q - L*D*U)/norm(A);
myError2 = norm(b-A*x)/norm(b);

fprintf('The error of ||PA-LU||/||A|| is %e\nThe error of ||b-Ax||/||b|| is %e\n',myError1,myE
```

2.1.3 Question 6

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 3
%
% Question
%   Problem 6
%
% Function Dependencies
%   cholDecomp.m
%
%
% Notes
%   The purpose of the question is unclear to me
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc

A = delsq(numgrid('S',5));
L = cholDecomp(A);
disp(L)
```

2.2 Accompanying Functions

2.2.1 Cholesky Decomposition

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHOLDECOMP.m
%
% DESCRIPTION
%   Decomposes a PSD matrix into its cholesky decomposition
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN PSD matrix
%
% OUTPUT
%   A - lower triangular matrix s.t GG^T = A
%
% NOTES
%   Outputs a LOWER triangular matrix
%   Would like to remove nested for loop (line 32)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%



function [A] = cholDecomp(A)

[~,n] = size(A);

for k = 1:n-1
    A(k,k) = sqrt(A(k,k));
    A(k+1:n,k) = A(k+1:n,k)/A(k,k);
    for j = k+1:n
        A(j:n,j) = A(j:n,j) - A(j,k)*A(j:n,k);
    end
end
A(n,n) = sqrt(A(n,n));
A = tril(A);
end
```

2.2.2 Forward Substitution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% FORWARDSUB.m
%
% DESCRIPTION
%   Solves Ax = b using forwards substituion
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%   b - Nx1 vector
%
% OUTPUT
%   x - solution to Ax = b
%
% NOTES
%   Asserts the size of matrix/vector as well as upper triangularity
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = forwardSub(A,b)

sizeA = size(A);
sizeB = size(b);

if sizeB(1) < sizeB(2)
    b = b';
end

n = sizeA(1);

x = zeros(n,1);
x(n) = b(n)/A(n,n);

for k = 1:n
    x(k) = (b(k) - A(k,1:k-1)*x(1:k-1))/A(k,k);
end

end
```

2.2.3 Backward Substitution

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% BACKSUB.m
%
% DESCRIPTION
%   Solves Ax = b using backwards substituion
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%   b - Nx1 vector
%
% OUTPUT
%   x - solution to Ax = b
%
% NOTES
%   Asserts the size of matrix/vector as well as lower triangularity
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = backSub(A,b)

sizeA = size(A);
sizeB = size(b);

if sizeB(1) < sizeB(2)
    b = b';
end


n = sizeA(1);
x = zeros(n,1);
x(n) = b(n)/A(n,n);

for k = n-1:-1:1
    x(k) = (b(k) - A(k,k+1:n)*x(k+1:n))/A(k,k);
end

end
```

2.2.4 Matrix Equation Solver (with or without pivoting)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BACKSUB.m
%
% DESCRIPTION
%   Solves Ax = b using LU factorization with optional partial pivoting
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%   b - Nx1 vector
%
% OUTPUT
%   x - solution to Ax = b
%
% NOTES
%   Asserts the matrix is square and the vector is of appropriate length
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x] = matSolve(A,b,~)
%% Assertions
n = size(A);
assert(n(1) == n(2))

assert(isvector(b));
assert(length(b) == n(1));

n = n(1);

%% Solving Ax = b
if nargin > 2
    [A,p] = pluFact(A,'pivot'); %factorize with pivoting
else
    [A,p] = pluFact(A); %factorize without pivoting
end
b = b(p); %repermute

diagonal = diag(A);
A(1:n+1:n^2) = ones(1,n); %change the diagonal in the combined LU matrix
y = forwardSub(A,b); %solve Ly = b
A(1:n+1:n^2) = diagonal;
```

```
x = backSub(A,y); %solve Ux = y


end
```

2.2.5 Extraction of P,L, and U Matrices

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% pluExtract.m
%
% DESCRIPTION
%   Returns the P,L,and U matrices from the pluFact return matrix
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix that is output from pluFact
%   p - Nx1 matrix that is output from pluFact
%
% OUTPUT
%   P - partial pivoting matrix
%   L - lower triangular matrix
%   U - upper triangluar matrix
%
% NOTES
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

function [P,L,U] = pluExtract(A,p)
n = length(A);

U = triu(A);
L = tril(A);
L(1:n+1:n^2) = ones(1,n);

P = eye(n);
for i = 1:n
    P(i,i) = 0;
    P(i,p(i)) = 1;
end
end
```

2.2.6 PLU Factorization

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% pluFact.m
%
% DESCRIPTION
%   Decomposes a matrix A using LU factorization with optional partial pivoting.
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   A - NxN matrix
%
% OUTPUT
%   A - Decomposed L/U matrix for efficient space storage
%   P - pivoting vector
%
% NOTES
%   Asserts the size of the input matrix
%   Updating the submatrix was taken directly from the textbook
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

function [A,p] = pluFact(A,~)

[m,n] = size(A);
assert(m == n,'Please insert square matrix')
p = 1:n;

for i = 1:n-1
    J = i+1:n;
    if nargin > 1
        %First, find the maximum row to swap with
        [~,loc] = max(abs(A(i:n,i)));
        loc = loc + i-1;

        %Swap row i with row loc and also in p
        A([i,loc],:) = A([loc,i],:);
        p([i,loc]) = p([loc,i]);
    end

    %Now replace the "eliminated" elements with their elimination constant
    A(J,i) = A(J,i) / A(i,i);

    %And then do the elimination on the rest of the matrix
```

```
    A(J,J) = A(J,J) - A(J,i) * A(i,J);
end
end
```