Assignment 2
SOLUTIONS

1. See the attached MATLAB code and comments.

2. a)

$$A = \begin{bmatrix} 1.000\times10^{-3} & 2.000\times10^{0} & 3.000\times10^{0} \\ -1.000\times10^{0} & 3.712\times10^{0} & 4.623\times10^{0} \\ -2.000\times10^{0} & 1.072\times10^{0} & 5.072\times10^{0} \end{bmatrix}$$

Step 1: $R2 + 1000R1 \to R2$, $R3 + 2000R1 \to R3$ $\Rightarrow$ $M_1 = \begin{bmatrix} 1.000 & & \\ 1.000\times10^{3} & 1.000 & \\ 2.000\times10^{3} & & 1.000 \end{bmatrix}$

$$A^{(1)} = M_1 A = \begin{bmatrix} 1.000\times10^{-3} & 2.000\times10^{0} & 3.000\times10^{0} \\ 0.000 & 2.004\times10^{3} & 3.005\times10^{3} \\ 0.000 & 4.001\times10^{3} & 6.005\times10^{3} \end{bmatrix}$$

Note that, I used rounding for approximating the value in this floating point system.

Step 2: $R3 - \frac{4.001}{2.004} R2 \to R_3$ $\Rightarrow$ $M_2 = \begin{bmatrix} 1.000 & & \\ & 1.000 & \\ & -1.997 & 1.000 \end{bmatrix}$

$$A^{(2)} = M_2 A^{(1)} = \begin{bmatrix} 1.000\times10^{-3} & 2.000\times10^{0} & 3.000\times10^{0} \\ 0.000 & 2.004\times10^{3} & 3.005\times10^{3} \\ 0.000 & 8.880\times10^{-1} & 4.015\times10^{0} \end{bmatrix}.$$

Observe that, $A^{(2)}$ is not upper triangular as expected. This can be solved by separating $M_2 = M_{21} M_{22}$ s.t.

$$M_{21} = \begin{bmatrix} 1.000 & & \\ & 1.000 & \\ & -4.001\times10^{3} & 1.000 \end{bmatrix} \quad M_{22} = \begin{bmatrix} 1.000 & & \\ & (2.004\times10^{3})^{-1} & \\ & & 1.000 \end{bmatrix}$$

and defining $A^{(2)} = M_{21}(M_{22}A^{(1)})$ (of course, only to a level, due to floating point operations getting exactly lower and upper triangular matrices is tricky)

b) If we compare part (a) to result obtained from attached LU without pivoting code – see it at the last pages –

(part a)

$$U = \begin{bmatrix} 0.001 & 2.000 & 3.000 \\ 0.000 & 2004 & 3005 \\ 0.000 & 0.988 & 4.015 \end{bmatrix} \quad L = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ -1000 & 1.000 & 0.000 \\ -2000 & 1.997 & 1.000 \end{bmatrix}$$

LU without pivoting

$$U = \begin{bmatrix} 0.001 & 2.000 & 3.000 \\ 0.000 & 2004 & 3005 \\ 0.000 & 0.000 & 5.922 \end{bmatrix} \quad L = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ -1000 & 1.000 & 0.000 \\ -2000 & 1.997 & 1.000 \end{bmatrix}$$

Lower triangular parts match up to the 4-digit precision but obviously upper triangular parts are quite different.

c) By not using pivoting in both cases, we removed it as a point of error (Note that, it is a point of error. In first step, we should pivot last ~~column~~ row entry to first ~~column~~ row) So all the errors we see, are coming from floating point operations. One is a cancellation error due to not pivoting and the other one is approximation of $\frac{4.001}{2.004}$.

3. For partial pivoting, we compare current diagonal entry to those all below it in the same column. So # comparisons:

$$\sum_{i=1}^{n-1}(n-i) = n(n-1) - \frac{n(n-1)}{2}$$

$$= + \frac{n(n-1)}{2} \approx O(n^2)$$

For complete pivoting, diagonal entry a_{ii} is compared to all other entries in submatrix $\begin{bmatrix} a_{ii} & \cdots & a_{in} \\ \vdots & \ddots & \vdots \\ a_{ni} & \cdots & a_{nn} \end{bmatrix}$. Hence

comparisons:

$$\sum_{i=1}^{n-1}(n-i)^2 = \frac{1}{6}\left(2n^3 - 3n^2 + n\right) \approx O\left(n^3\right)$$

4. $\boxed{\Rightarrow}$ Assume A has LU factorization. Divide A as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11} is a square matrix of size k, $1 \le k \le n$. Divide L and U in the same way.

$$L = \begin{bmatrix} L_{11} & L_{12}=0 \\ L_{21} & L_{22} \end{bmatrix}, \quad U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21}=0 & U_{22} \end{bmatrix}.$$

Now since $A = LU$, $A_{11} = L_{11}U_{11}$. Consider

$\det(A_{11}) = \det(L_{11} U_{11}) = \det(L_{11}) \det(U_{11})$. We know that $\det(L_{11}) \neq 0$ and $\det(U_{11}) \neq 0$, hence, $\det(A_{11}) \neq 0$. Therefore A_{11} is non singular.

$\boxed{\Leftarrow}$ Remind that, a non-singular matrix A has an LU factorization iff all its leading principal minors (pivots) are nonzero. Assume each $A_{1:k,1:k}$ is non-singular. Before going further, let us make some observations. Take M, square matrix with its LU factorization L_M, U_M. Pivots of M are $U_M(k,k)$. $\det(U_M) = \prod_k U_M(k,k)$. From this we can deduce each

$U_m(k,k) = \det(U_m(1:k,1:k)) / \det(U_m(1:k-1,1:k-1))$. In addition to this, $\det(M) = \det(U_M)$, thus, $U_M(k,k) = \det(M(1:k,1:k)) / \det(M(1:k-1,1:k-1))$.

Using this observation, since each $A(1:k,1:k)$ is non-singular $\det(A(1:k,1:k)) \neq 0$. Hence each pivot of A is nonzero, and we know that A is nonsingular. Therefore, there is an LU factorization for A.

5) First soln

a,b) Such an algorithm is Gauss-Jordan elimination and its cost is $O(n^3)$.

c) Each step of the algorithm can be thought as matrix-matrix multiplications, i.e.

$$M * [A \mid I].$$

But M is mostly zero, except some off diagonal entries and all diagonal entries. Hence by skipping arithmetic operations with zero entries, we can improve the algorithm.

See the second solution at the end!

6) Cost of solving a tridiagonal system of equations is $O(n)$ with the given algorithm. Cost of inverting the same matrix is $O(n^3)$ and it is not guaranteed that tridiagonal structure of the matrix will be preserved (i.e. memory concerns).

7) Consider $A^{(1)}$, the matrix we get after one step of Gaussian elimination. Entries of the matrix $A^{(1)}$ satisfies;

$$a_{ij}^{(1)} = a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} \cdot a_{1j}^{(0)}$$

assuming we are using partial pivoting, $\left|\frac{a_{i1}^{(0)}}{a_{11}^{(0)}}\right| \le 1$. So

$$|a_{ij}^{(1)}| \le |a_{ij}^{(0)}| + \left|\frac{a_{i1}^{(0)}}{a_{11}^{(0)}} \cdot a_{1j}^{(0)}\right| \le \underbrace{\qquad}_{|a_{ij}^{(0)}| + |a_{1j}^{(0)}|} \le 2\max_{i,j}\left(|a_{ij}^{(0)}|\right)$$

Now, $|u_{ij}| = |a_{ij}^{(n-1)}| \leq 2 \max_{i,j} |a_{ij}^{(n-2)}|$

$$\leq ---$$

$$\leq 2^{n-1} \max_{i,j} |a_{ij}^{(0)}|$$

$$\Rightarrow \max_{i,j} |u_{ij}| \leq 2^{n-1} \max_{i,j} |a_{ij}^{(0)}| \quad , \quad a_{ij}^{0} = a_{ij}$$

$$\Rightarrow \rho \leq 2^{n-1}$$

8) MATLAB implementation is faster by a factor of 256. This is expected as MATLAB's lu is compiled, but implementation attached is interpreted. Naively, ~~do~~ the complexity of LU factorization is $O(n^3)$. My CPU is Intel Core i5-4278U @ 2.60 GHz. It has two cores and four threads, but for this case parallel performance is irrelevant. Sequential theoretical flops is 3.84 GFLOPs according to the online sources. ~~Assuming~~ Hence, expected time is $\frac{\text{cost}}{\text{FLOPS}} = \frac{1000^3}{3.84 \times 10^9} \approx 0.26\text{s}$.

This might be, due to a lot of technical nuances; like size of the cache or pipelining, only accurate upto an order. But we can say my implementation is way slower (6.5 secs) and ~~MA~~MATLAB implementation is way faster (0.025 secs) compared to theoretical time.

For second part of the question, if size of matrix doubles, time should scale by an order of 8.

n	MATLAB LU	LU
100	0.0008	0.0089
200	0.0032	0.0159
400	0.0033	0.1181
800	0.0120	2.3210

Rate for MATLAB is around 2, i.e. if size doubles time is multiplied by 4. Rate for my implementation is around 4 at the last level. Neither catches the expectations. This might be due to, matrix sizes being too small so we do not see the real trend, my code being non-optimized, timing errors due to short computation time, ... If you are interested in more computational aspects of linear algebra check;

apfel.mathematik.uni-ulm.de/~lehn

especially FLENS and Tutorial for High-Performance GEMM.

```
function q8()
for i=1:5

  rng(1) % To guarantee reproducibility

  if i==1
      n = 1000;
  elseif i==2
      n = 100;
  else
      n = 2*n;
  end
  A = rand(n);

  tic
  [L,U] = lu(A);
  t_matlab=toc;

  tic
  [L,U,P] = lu_pivot(A);
  t_my=toc;

  fprintf('For A=rand(%4d);\n',n);
  fprintf('\ttime for MATLAB lu = %.4f\n',t_matlab);
  fprintf('\ttime for my implementation = %.4f\n',t_my);

end
```

```
function [L, U, P] = lu_pivot(A)
[n, ~] = size(A); % Obtain number of rows (should equal number of columns)
L=eye(n); P=eye(n); U=A; % Initialize the matrices
for j = 1:n
  [~, m] = max(abs(U(j:n, j))); % find the position of abs max in the column
  m = m+j-1; % shift due to implementation choices
  if m ~= j
    U([m,j],:) =  U([j,m], :);   % interchange rows m and j in U
    P([m,j],:) =  P([j,m], :);   % interchange rows m and j in P
    if j >= 2
      L([m,j],1:j-1) =  L([j,m], 1:j-1);
    end;
  end
  for i = j+1:n
    % For each row i, access columns from j+1 to the end and divide by
    % the diagonal coefficient at A(i ,i)
    L(i, j) = U(i, j) / U(j, j);

    % For each row i+1 to the end, perform Gaussian elimination
    % In the end, U will become upper triangular
    U(i, :) =  U(i, :) - L(i, j)*U(j, :);
  end
end
```

```
function [L, U] = lu_nopivot(A)
n = size(A, 1); % Obtain number of rows (should equal number of columns)
L = eye(n); % Start L off as identity and populate the lower triangular half
            % slowly
for k = 1 : n
    % For each row k, access columns from k+1 to the end and divide by
    % the diagonal coefficient at A(k ,k)
    L(k + 1 : n, k) = A(k + 1 : n, k) / A(k, k);
```

```
        % For each row k+1 to the end, perform Gaussian elimination
        % In the end, A will contain U
        for l = k + 1 : n
            A(l, :) = A(l, :) - L(l, k) * A(k, :);
        end
    end
    U = A;
```

5) Second Solution

a) Consider $AX = I$ where $X \in \mathbb{R}^{n\times n}$ and I is the identity of size n. Then, by considering each column x_i of X and e_i of I, problem reduces to $Ax_i = e_i$.

Algorithm: – $[L,U] = lu(A)$;
– for $i = 1:n$
solve $Ly = I(:,i)$;
solve $UX(:,i) = y$;
– end for

b) Cost of Algorithm = cost of LU factorization
+ n times forward substitution
+ n times backward substitution
$= O(n^3) + n\,O(n^2) + n\,O(n^2)$
$= O(n^3)$

c) Observe that for $Ly = b$, if $b = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b_{k-1} \\ \vdots \\ b_n \end{bmatrix}$, i.e. first few entries are zero then $y = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ y_{k-1} \\ \vdots \\ y_n \end{bmatrix}$. So for this step in the algorithm, we can reduce the cost by;

in place of "solve $Ly = I(:,i)$" put set $y(1:i) = 0$
solve $L(i+1:end, i+1:end)\,y(i+1:end) = I(i+1:end, i)$.