1 Problems

1. To solve the system of equations given by:

\[\begin{align*}
    x + 2y - z &= 2 \\
    3y + z &= 4 \\
    2x - y + z &= 2
\end{align*}\]

with Gaussian elimination we begin by putting the entire equation into a \(3 \times 4\) matrix:

\[
\begin{pmatrix}
    1 & 2 & -1 & 2 \\
    0 & 3 & 1 & 4 \\
    2 & -1 & 1 & 2
\end{pmatrix}
\]

Then we proceed to carry out elementary row operations. First we add \(-\frac{1}{2}\) row 1 to row 3 to get. For ease of notation we will write this as \(r_3 = r_3 - \frac{1}{2}r_1\). This gives:

\[
\begin{pmatrix}
    1 & 2 & -1 & 2 \\
    0 & 3 & 1 & 4 \\
    0 & 0 & 1 & 1
\end{pmatrix}
\]

next, \(r_1 = r_1 + r_3\) and \(r_2 = r_2 + r_1\):

\[
\begin{pmatrix}
    1 & 2 & 0 & 3 \\
    0 & 3 & 0 & 3 \\
    0 & 0 & 1 & 1
\end{pmatrix}
\]
and finally $r_1 = r_1 - \frac{2}{3}r_2$ and $r_2 = \frac{1}{3}r_2$:

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The we see the solution to this system is $x = 1, y = 1, z = 1$. We can apply a similar process to solve:

\[
\begin{align*}
2x + y - 4z &= -7 \\
x - y + z &= -2 \\
-x + 3y - 2z &= 6
\end{align*}
\]

we write it as a matrix:

$$\begin{pmatrix} 2 & 1 & -4 & -7 \\ 1 & -1 & 1 & -2 \\ -1 & 3 & -2 & 6 \end{pmatrix}$$

then $r_2 = r_2 - \frac{1}{2}r_1$ and $r_3 = r_3 + \frac{1}{2}r_1$ gives us:

$$\begin{pmatrix} 2 & 1 & -4 & -7 \\ 0 & -3/2 & -1 & 3/2 \\ 0 & 7/2 & -4 & 5/2 \end{pmatrix}$$

then $r_1 = \frac{1}{2}r_1$ and $r_2 = -\frac{3}{2}r_2$:

$$\begin{pmatrix} 1 & 1/2 & -2 & -7/2 \\ 0 & 1 & -2 & 1 \\ 0 & 7/2 & -4 & 5/2 \end{pmatrix}$$

next $r_3 = r_3 - \frac{7}{2}r_2$

$$\begin{pmatrix} 1 & 1/2 & -2 & -7/2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 3 & 6 \end{pmatrix}$$

then $r_3 = \frac{1}{3}r_3$
\[
\begin{pmatrix}
1 & 1/2 & -2 & -7/2 \\
0 & 1 & -2 & 1 \\
0 & 0 & 1 & 2 \\
\end{pmatrix}
\]

next \(r_1 = r_1 + 2r_3\) and \(r_2 = r_1 + 2r_3\)

\[
\begin{pmatrix}
1 & 1/2 & 0 & 1/2 \\
0 & 1 & 0 & 3 \\
0 & 0 & 1 & 2 \\
\end{pmatrix}
\]

and finally: \(r_1 = r_1 - \frac{1}{2}r_2\) gives us:

\[
\begin{pmatrix}
1 & 0 & 0 & -1 \\
0 & 1 & 0 & 3 \\
0 & 0 & 1 & 2 \\
\end{pmatrix}
\]

give us that \(x = -1, y = 3, z = 2\).

2. If we know the approximate operation count for Gaussian elimination is \(\frac{2n^3}{3}\) and that it taxes 1 second to solve a \(n \times n\) system then it will take roughly \(4^3 = 64\) seconds to compute a \(4n \times 4n\) matrix.

One way to think about this is with a toy example. Let \(n = 1\), then if the algorithm acting on a \(1 \times 1\) matrix takes 1 second the computer preforms \(\frac{2}{3}\) operations per second (yes this is silly, but it’s a thought experiment). Then a \(4 \times 4\) matrix will require \(\frac{2 \times 4^3}{3} = \frac{128}{3}\) operations, and if the computer can compute \(\frac{2}{3}\) operations per second then it will take \(\frac{128}{3} \div \frac{2}{3} = 64\) seconds to complete.

3. To compute the LU factorization of the matrix:

\[
A = \begin{pmatrix}
4 & 2 & 0 \\
4 & 4 & 2 \\
2 & 2 & 3 \\
\end{pmatrix}
\]

we will multiply by elementary matrices. We begin with \(r_2 = r_2 - r_1\)

\[
\begin{pmatrix}
1 & 0 & 1 \\
-1 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix} \begin{pmatrix}
4 & 2 & 0 \\
4 & 4 & 2 \\
2 & 2 & 3 \\
\end{pmatrix} = \begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
2 & 2 & 3 \\
\end{pmatrix}
\]

next \(r_3 = r_3 - \frac{1}{2}r_1\)
\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
-1/2 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
2 & 2 & 3
\end{pmatrix}
= 
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
0 & 1 & 3
\end{pmatrix}
\]
and finally \( r_3 = r_3 - \frac{1}{2}r_2 \)
\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & -1/2 & 1
\end{pmatrix}
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
0 & 1 & 3
\end{pmatrix}
= 
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
0 & 0 & 2
\end{pmatrix}
\]

hence we have found \( U \). Now we multiply all of the elementary matrices together and call the product \( E \):
\[
E = 
\begin{pmatrix}
1 & 0 & 0 \\
-1 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
-1/2 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & -1/2 & 1
\end{pmatrix}
= 
\begin{pmatrix}
1 & 0 & 0 \\
-1 & 1 & 0 \\
-1/2 & -1/2 & 1
\end{pmatrix}
\]

So now we’ve found \( EA = U \) and our goal is to find \( A = LU \). Then we can observe \( L = E^{-1} \) however, rather than computing the inverse of \( E \) directly, we can note that it is lower triangular with ones on the diagonals, so its inverse is given by switching the signs on the entries below the diagonal. Hence:
\[
L = \begin{pmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
1/2 & 1/2 & 1
\end{pmatrix}.
\]

We can check this by computing:
\[
LU = 
\begin{pmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
1/2 & 1/2 & 1
\end{pmatrix}
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
0 & 0 & 2
\end{pmatrix}
= 
\begin{pmatrix}
4 & 2 & 0 \\
4 & 4 & 2 \\
2 & 2 & 3
\end{pmatrix}
= A.
\]

4. If we wish to solve the problem:
\[
\begin{pmatrix}
4 & 2 & 0 \\
4 & 4 & 2 \\
2 & 2 & 3
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= 
\begin{pmatrix}
2 \\
4 \\
6
\end{pmatrix}
\]

with LU factorization and double back substitution we can think of the problem as solving \( LUx = b \) by first solving \( Ly = b \) then \( Ux = y \). Then our first back substitution problem is:
\[
\begin{pmatrix}
1 & 0 & 0 \\
1 & 1 & 0 \\
1/2 & 1/2 & 1
\end{pmatrix}
\begin{pmatrix}
y_1 \\
y_2 \\
y_3
\end{pmatrix}
= 
\begin{pmatrix}
2 \\
4 \\
6
\end{pmatrix}
\]
or:

\[
y_1 = 2 \\
y_1 + y_2 = 4 \\
\frac{1}{2}y_1 + \frac{1}{2}y_2 + y_3 = 6
\]

Then clearly \(y_2 = 2\), from the second equation we find \(2 + y_2 = 4\) so \(y_2 = 2\). Plugging these values into the third equation we find: \(\frac{1}{2}2 + \frac{1}{2}2 + y_3 = 6\) so \(y_3 = 4\). Now that we have solved for \(y\) we can set up and solve our second back substitution problem:

\[
\begin{pmatrix}
4 & 2 & 0 \\
0 & 2 & 2 \\
0 & 0 & 2
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3
\end{pmatrix}
= 
\begin{pmatrix}
2 \\
2 \\
4
\end{pmatrix}
\]

From the third equation we see \(x_1 = 2\). Plugging this into the second equation we see \(x_2 = -1\), and then plugging this into the first equation we find \(x_1 = 2\). We can check this by verifying the matrix multiplication:

\[
\begin{pmatrix}
4 & 2 & 0 \\
4 & 4 & 2 \\
2 & 2 & 3
\end{pmatrix}
\begin{pmatrix}
1 \\
-1 \\
2
\end{pmatrix}
= 
\begin{pmatrix}
2 \\
4 \\
6
\end{pmatrix}
\]

5. Code to compute the LU factorization of a matrix and preform back substitution to compute \(Ax = b\) is attached. (Code by Prof. Banks). A typical run find that for a 100 \(\times\) 100 matrix: \(\text{norm}(A-L*U) \approx 7.7e-13\) and \(\text{norm}(A\mathbf{x} - \mathbf{b}) \approx 2.6e-12\) and solve times look like:

\begin{tabular}{l l}
N=100: & tFact=2.542061e-02, tSolve=5.571113e-04 \\
N=200: & tFact=1.413687e-01, tSolve=1.457850e-03 \\
N=400: & tFact=1.138721e+00, tSolve=6.783867e-03 \\
N=800: & tFact=1.063505e+01, tSolve=3.163177e-02
\end{tabular}
From these results we observe that most of the time spent in this process is from the factorization of the matrix. We also observe that the costs for the factorizations grow roughly like $n^3$ by computing:

$$\frac{\log(t_{Fact}(2N)/t_{Fact}(N))}{\log(2)}$$

(here log indicates the natural log) which gives us:

- $N = 200$: Observed OrdFact = $O(2.90)$
- $N = 400$: Observed OrdFact = $O(3.02)$
- $N = 800$: Observed OrdFact = $O(3.50)$
- $N = 1600$: Observed OrdFact = $O(3.39)$

Preforming the same calculations for the solve times gives us:

- $N = 200$: Observed OrdSolve = $O(2.02)$
- $N = 400$: Observed OrdSolve = $O(2.32)$
- $N = 800$: Observed OrdSolve = $O(2.49)$
- $N = 1600$: Observed OrdSolve = $O(2.21)$

which suggests (but does not prove) that the costs of the substitution steps are roughly $n^2$.

## 2 Codes

```matlab
%% test of LU codes
N = 100;
A = rand(N,N);
b = rand(N,1);

%% run my LU factorization
[L,U] = my_lu( A );

fprintf('check factorization
');
fprintf(' norm(A-L*U)=%e
',norm(A-L*U) );

x = my_lu_solve( L,U,b );
fprintf('check solution
');
fprintf(' norm(Ax-b)=%e
',norm(A*x-b) );
```
for i=0:4 % only need to go to 2 for assignment, but we'll do more to experiment
    N = 100*2ˆi;
    A = rand(N,N);
    b = rand(N,1);

    tic
    [L,U] = my_lu( A );
    tFact(i+1) = toc;

    tic
    x = my_lu_solve( L,U,b );
    tSolve(i+1) = toc;

    fprintf( 'N=%i: tFact=%e, tSolve=%e
',N,tFact(i+1),tSolve(i+1) );
end

%% code by Stefan
% find and print fact orders
for i=2:length(tSolve)
    ordFact = (log(tFact(i)/tFact(i-1)))/log(2);
    fprintf( 'N = : %i Observed OrdFact = O(%1.2f) \n',100*2ˆ(1-1),ordFact);
end

% find and print solve orders
for i=2:length(tSolve)
    ordSolve = (log(tSolve(i)/tSolve(i-1)))/log(2);
    fprintf( 'N = : %i Observed OrdSolve = O(%1.2f) \n',100*2ˆ(1-1),ordSolve);
end

function [L,U] = my_lu( A );

    [n,m] = size( A );

    if( n ~= m )
        fprintf( 'ERROR: matrix not square\n' );
        stop
    end

    L = eye( n,n );
    U = A;

    for k = 1:n-1
        if( abs(U(k,k)) < 10*eps )
            fprintf( 'ERROR: zero pivot element\n' );
            stop
        end

        for i = k+1:n
            m = U(i,k)/U(k,k);
L(i,k) = m;
for j = k:n
    U(i,j) = U(i,j) - m*U(k,j);
end
end

return

function x = my_lu_solve( L,U,b )

[n,m] = size( L );

if( n ~= m )
    fprintf( 'ERROR: matrix not square\n' );
    stop
end

%% solve Ly=b
y = b;
for i = 2:n
    for j = 1:i-1
        y(i) = y(i) - L(i,j)*y(j);
    end
end

%% solve Ux=y
x = y;
x(n) = x(n) / U(n,n);
for i = n-1:-1:1
    for j = i+1:n
        x(i) = x(i) - U(i,j)*x(j);
    end
    x(i) = x(i) / U(i,i);
end