SE 284 (AUG) 2:1 Numerical Linear Algebra
Homework 1

Submitted by: Aakriti Gupta
06-02-01-10-51-12-1-09237

Generate a random matrix, A, of dimension nxn and a random vector, b, of dimension nx1, with n being a positive integer (>=2). To make A diagonally dominant, add 100 to the diagonal entries of A. We are interested in solving Ax = b.

```
function New(n)

if n < 2
    exit();
end

A = randn(n);
b = randn(n,1);
for i=1:n
    A(i,i) = A(i,i) + 100;
end
```

Use Gaussian Elimination to find x for n = 100. Use the command x = A\b in MATLAB and compare the obtained solution with the Gaussian elimination solution. Plot the difference in the solution (relative error), can you explain the difference?

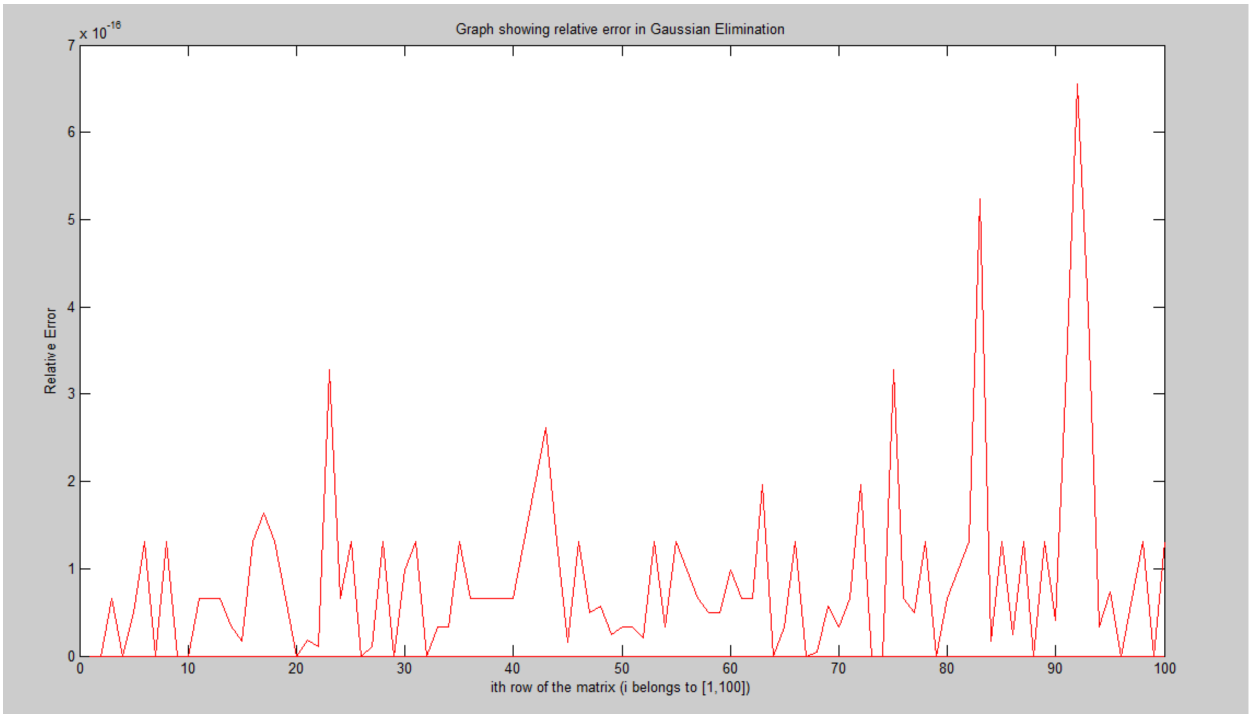
```
function x = Gaussian(A,b,n)

Z = [A b];

for i = 1:n-1
    for k = i+1 : n
        T = -Z(k,i)/Z(i,i);
        for j = i:n+1
            Z(k,j) = T*Z(i,j) + Z(k,j);
        end
    end
end

x = zeros(n,1);
c = Z(:,n+1);
Z = Z(:,1:n);

x(n) = c(n)/Z(n,n);
for i = n-1:-1:1
    x(i) = c(i);
    for j = i+1:n
        x(i) = x(i) - Z(i,j)*x(j);
    end
    x(i) = x(i)/Z(i,i);
end
```

Now, code of the new function (that generates random matrices A & b) can be extended to call this Gaussian function and compute the relative error. The following lines are added.

```
y = A\b;
x = Gaussian(A,b,n);
E = abs(x-y)/abs(y);
plot(E,'r');
```

Command to see the results:

```
>> New(100)
```



Explanation of the difference:

Machine implementation of Gaussian method suffers from Round Off Error. This is more apparent in case of small pivots (nearly 0). Since the matrix is explicitly made diagonally dominant here, the error is relatively small, order of 10^(-16).

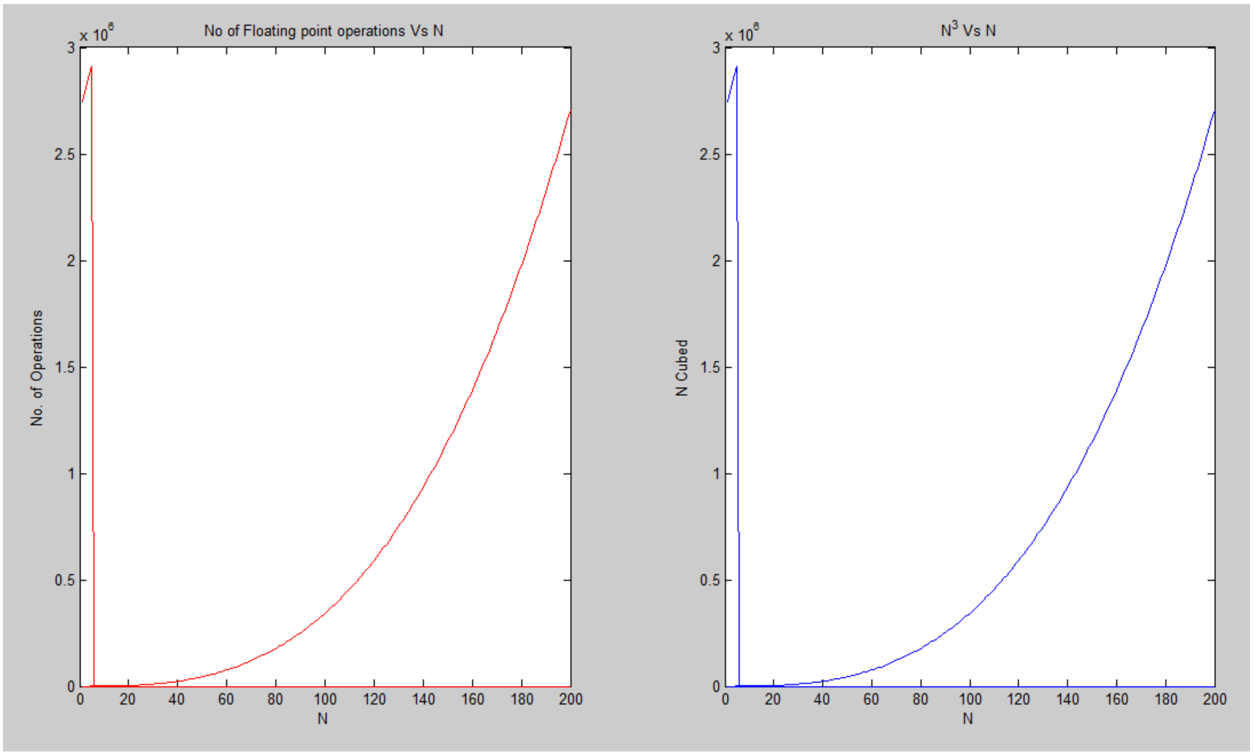
Can you verify that Gaussian Elimination order of computation is O(n3) through usage of your programs?

In function Gaussian, we take a count variable that counts the total number of floating point multiplications that occur during execution of algorithm.

```
Z = [A b];
count = 0;

for i = 1:n-1
    for k = i+1 : n
        T = -Z(k,i)/Z(i,i);
        count = count + 1;
        for j = i:n+1
            Z(k,j)  = T*Z(i,j)  + Z(k,j);
            count = count + 1;
        end
    end
end
```

Now calling this function for different values of n give different value of the count variable i.e. the number of floating point operations.
Note: The code for backward substitution is not taken into account as the order of computation is asymptotically lesser hence can be ignored.

For different values of N, the count value is stored in a vector. Next we plot N^3 and the value of the count vector. The resultant graph is:



Thus the no. of operations required are O(N^3).

Write LU decomposition for the same matrix. Find x.

```
function x = LU(A,b,n)

A1 = A;
L=eye(n);
U=eye(n);

%Computation of Compact Storage Matrix.
for i = 1:n-1
    for k = i+1 : n
        T = -A1(k,i)/A1(i,i);
        for j = i+1:n
            A1(k,j) = T*A1(i,j) + A1(k,j);
        end
        A1(k,i)= - T;
    end
end

%Separating into L and U matrices.
for i = 1:n
    for j = 1:n
        if i>j
            L(i,j) = A1(i,j);
        else
            U(i,j) = A1(i,j);
        end
    end
end

w = zeros(n,1);
w(1) = b(1);

%Forward Substitution L(UX)=B => LW=B => UX=W
for i = 2:n
    w(i) = b(i);
    for j = 1:i-1
       w(i) = w(i) - L(i,j) * w(j);
    end
end

%Backward Substitution => LW=B
x(n) = w(n)/U(n,n);
for i = n-1:-1:1
    x(i) = w(i);
    for j = i+1:n
        x(i) = x(i) - U(i,j)*x(j);
    end
    x(i) = x(i)/U(i,i);
end
```

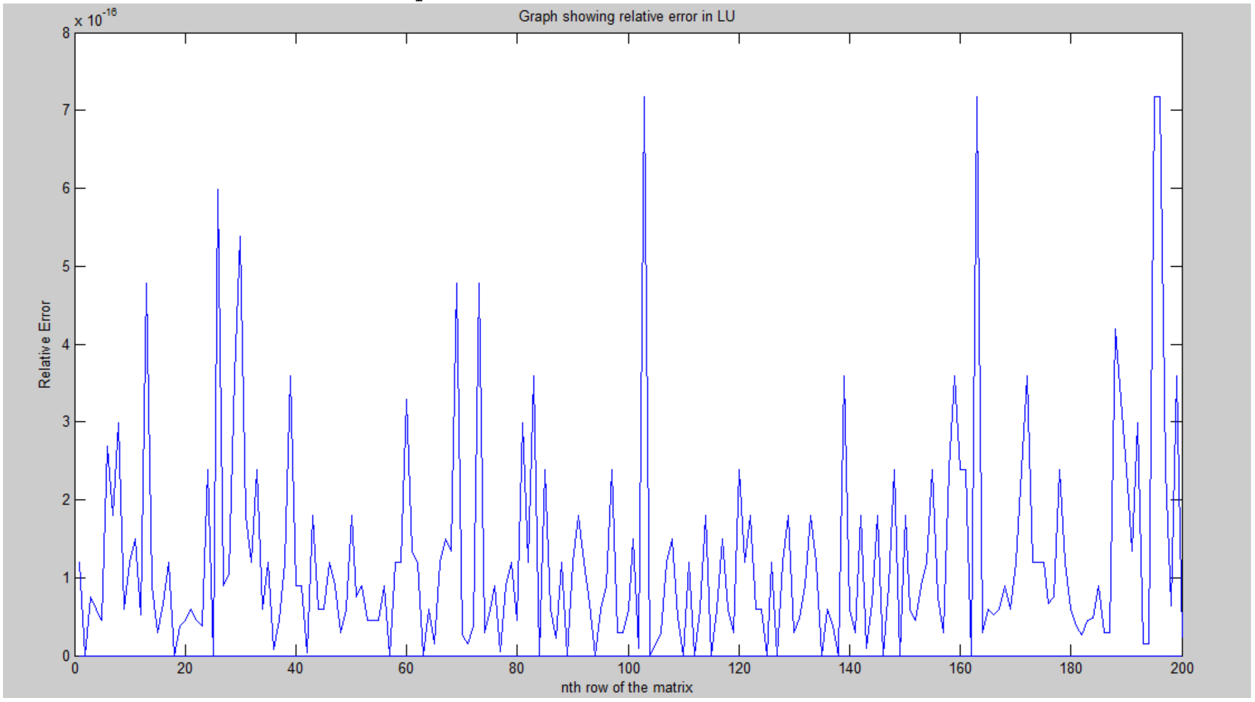
Repeat the same comparison of solution as in Gaussian for n = 200 using the programs written for LU, include additional comparison with Gaussian elimination solution. Explain the difference in the solutions obtained.

The code for comparison looks like this. It is added to the file that creates the random matrices A & b so that the same matrix is used for analyzing Gaussian as well as LU Decomposition.

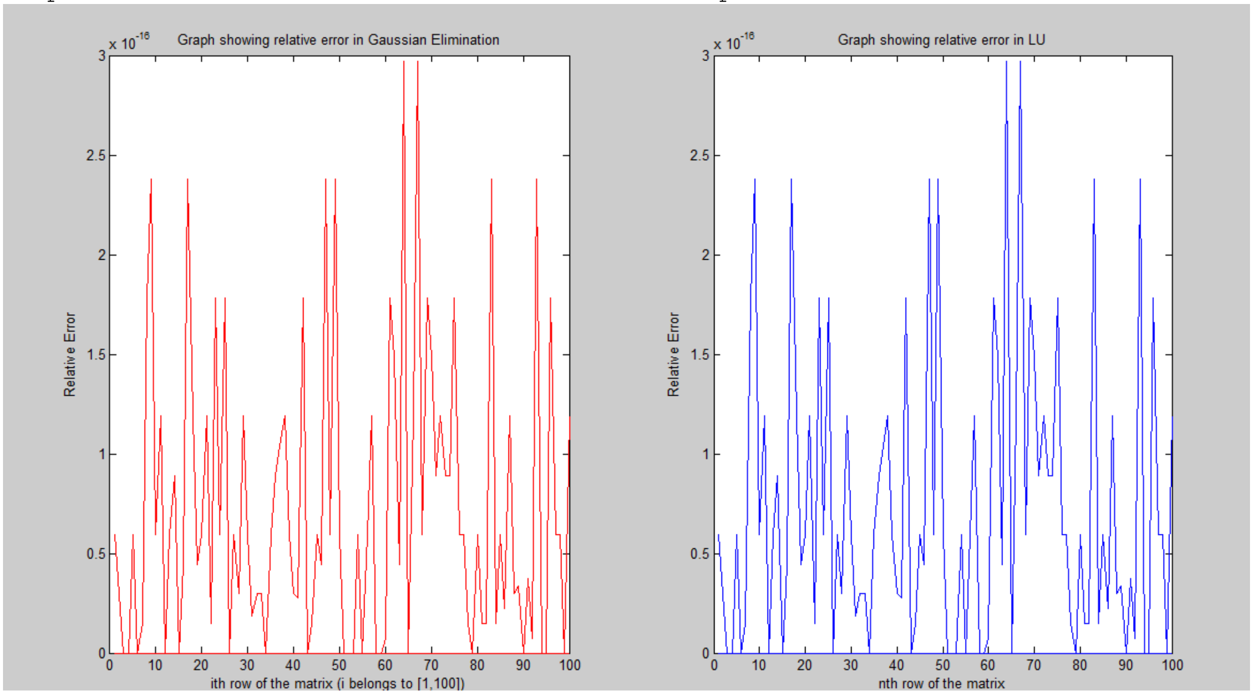
```
y = A\b;

tic
x = Gaussian(A,b,n);
t = toc
E = abs(x-y)/abs(y);

tic
x1 = LU(A,b,n);
t1 = toc
E1 = abs(x1-y)/abs(y);

subplot(1,2,1);
plot(E,'r');
title('Graph showing relative error in Gaussian Elimination');
xlabel('ith row of the matrix (i belongs to [1,100])');
ylabel('Relative Error');
hold on;
subplot(1,2,2);
plot(E1,'b');
title('Graph showing relative error in LU');
xlabel('nth row of the matrix');
ylabel('Relative Error');
```

Explanation: The relative error in both Gaussian and LU decomposition is same. The graphs coincide. Although the number of operations is different in these two cases and tic-toc command shows that Gaussian takes lesser time than LU decomposition.
The fact that both approaches have equal error could be attributed to round off.

```
Relative Error in LU Decomposition for n=200:
```

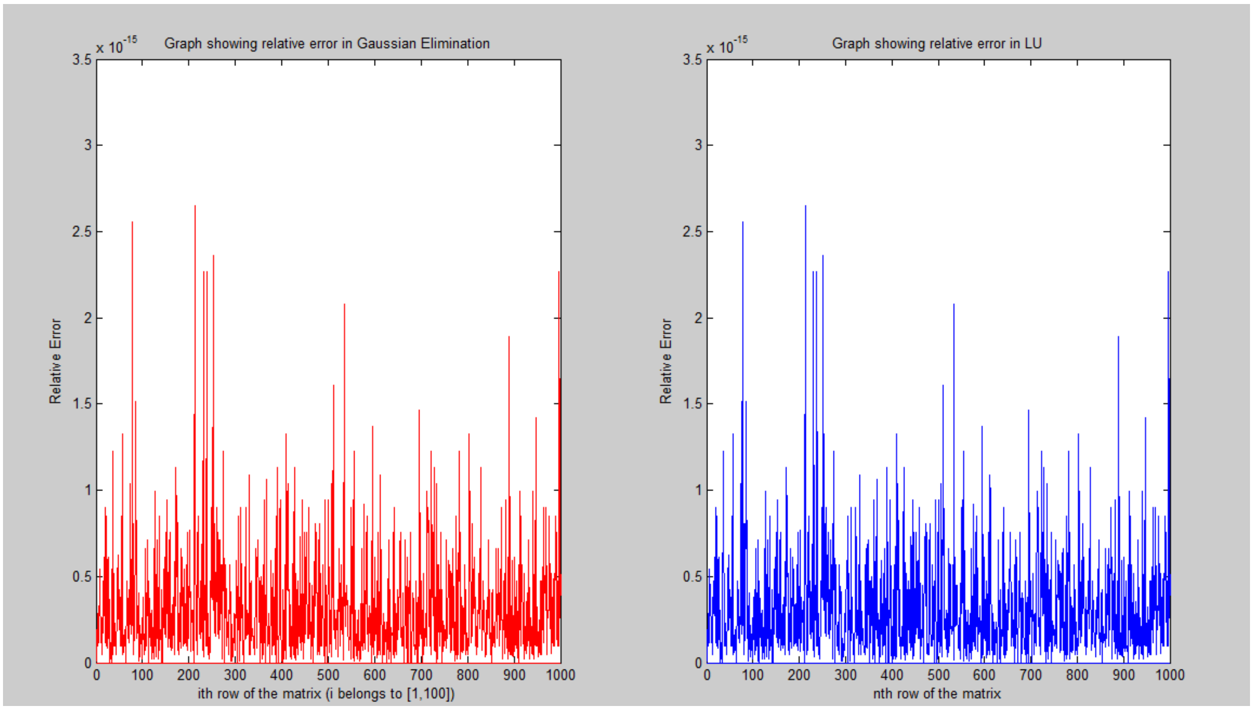


```
Comparison of Gaussian Elimination & LU Decomposition:
```



For n = 1000, find x using Gaussian Elimination and LU decomposition, which is computationally efficient? Time both algorithms and explain your findings.

For n = 1000 the difference can be seen.
We take note of the time only of the term of order n^3. Backward substitution and forward substitution are order n^2 operations and hence for larger n they are not decisive factors.
Gaussian elimination takes 17.3108 seconds & 334332000 operations and LU decomposition takes 17.3082 seconds & 333333000 operations. Hence Gaussian is slightly less efficient.


Graph showing relative error in Gaussian Elimination
x 10^-15
Relative Error
ith row of the matrix (i belongs to [1,100])
Graph showing relative error in LU
x 10^-15
Relative Error
nth row of the matrix