Report on

Assignment 1: Solving System of Linear Equations using Doolittle Algorithm LU Decomposition

Submitted in partial fulfillment of the requirements of
ENM 502
Numerical Methods and Modeling
by

Jalaj Maheshwari
M.S.E., Mechanical Engineering and Applied Mechanics

University of Pennsylvania
School of Engineering and Applied Sciences

16th February, 2016

Table of Contents

Table of Figures

1. Introduction

The problem statement of this particular project deals with formulating the Doolittle Algorithm to solve a system of liner equations using the LU-Decomposition with and without partial pivoting. The code for the same is developed on MATLAB (see Appendix) and various test cases are run to determine the validity of the developed code. The intrinsic MATLAB operator for LU decomposition for the said test cases is run to draw a comparison with the developed code. The general equation being solved here is represented by Equation 1.

$$Ax = B \tag{1}$$

Gaussian elimination with pivoting is the most efficient and accurate way to solve a linear system of equations. Most of the work in this method is spent on the matrix *A* itself. If we need to solve several different systems with the same A, and A is big, then we would like to avoid repeating the steps of Gaussian elimination on *A* for every different *B*. This can be accomplished by the LU decomposition, which in effect records the steps of Gaussian elimination.

The LU decomposition algorithm factors the matrix *A* into product of a lower and an upper triangular matrix, hence the 'LU' in the algorithm. There are various methods to factor matrix *A* into upper and lower triangular matrix. However, the Doolittle algorithm is particularly nice for two reasons. The first being that it can be easily programmed for even fairly complex storage arrangements for sparse matrices. The other reason is that the algorithm itself is a proof that the factorization is unique. Hence, the Doolittle algorithm is used to develop the code for LU decomposition for this particular project.

2. Problem Setup and Formulation

The Doolittle algorithm decomposes a non-singular n×n matrix into the product of an n×n lower triangular matrix and an n×n upper triangular matrix. Hence, after decomposition, matrix *A* can be represented as in Equation 2,

$$A = LU \tag{2}$$

where, L is the lower triangular matrix and U is the upper triangular matrix. Once this is done, the system of linear equations, now represented by Equation 3,

$$Ax = LUx = B \tag{3}$$

is solved by solving the system of linear equations now represented by Equation (4a) and (4b).

$$LD = B \tag{4a}$$

$$Ux = D \tag{4b}$$

Here, Equation (4a) is solved using forward substitution for and the subsequent Equation (4b) is solved for x using backward substitution. It is to be noted that LU decomposition can only be possible if the determinant of matrix *A* is non-zero.

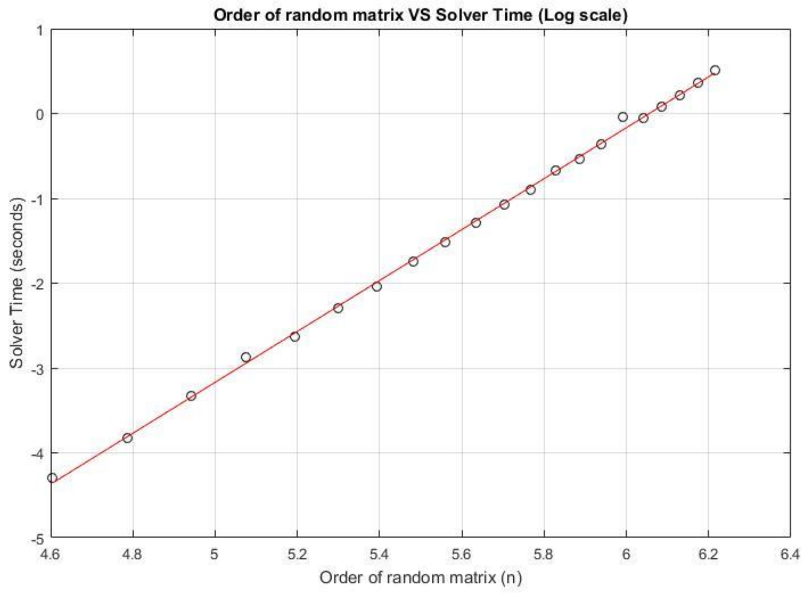
For Doolittle decomposition with pivoting, for each value of *i* from *k* to *n*, the maximum value of the magnitude of $a_{ik}^{(k)}$ is found and the rows are interchanged. Here, *i* is the row number, *k* is any column and *n* is the order of the square matrix. The algorithm then stores the switch information and proceeds as before. This method is called as partial pivoting, which has been implemented in the code. The rows are exchanged in a way to avoid zero at the diagonal positions in the matrix. For full pivoting, both the rows and the columns are searched for the maximum pivot element.

In this particular code, instead of switching the rows entirely, the indexing of the rows is changed and the algorithm is continued as is. This is less computationally intensive as compared to switching complete rows and hence speeds up the processing time. Various test cases are run corresponding to different types of matrices, with pivoting turned on and off. The tolerance in the code is set to 1e-14. The results and the discussion of the obtained results is provided in the next section.

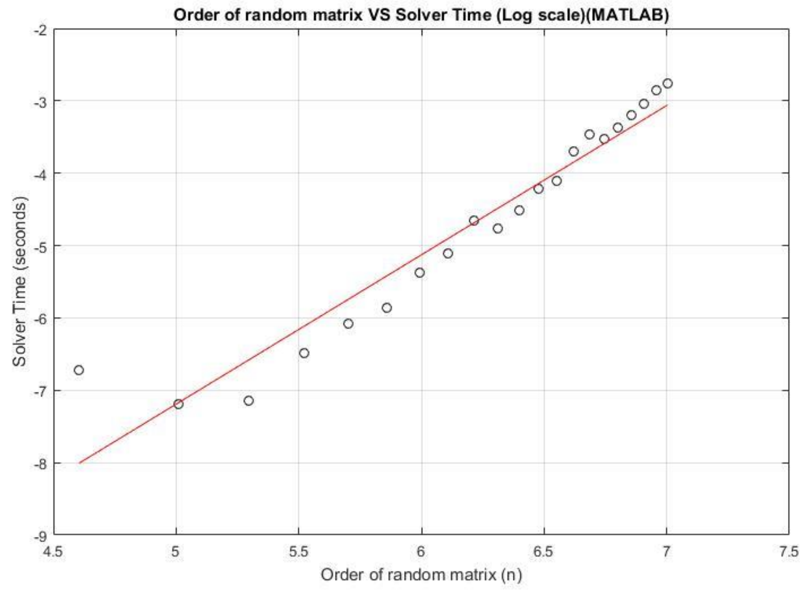
3. Results and Discussion

3.1 LU Decomposition for Random Matrix (Pivoting On)

For the first part of the project, the LU decomposition code is compared with the intrinsic MATLAB operator for the same for different orders of random matrices. Random values are assigned for matrix $A(n\times n)$ and $B(n\times 1)$. The computational time taken to decompose the matrix using both the code and the intrinsic operator is plotted against the order of the matrix to determine the scaling of decomposition of the two. Figure 1 shows the results for the same.



(a)



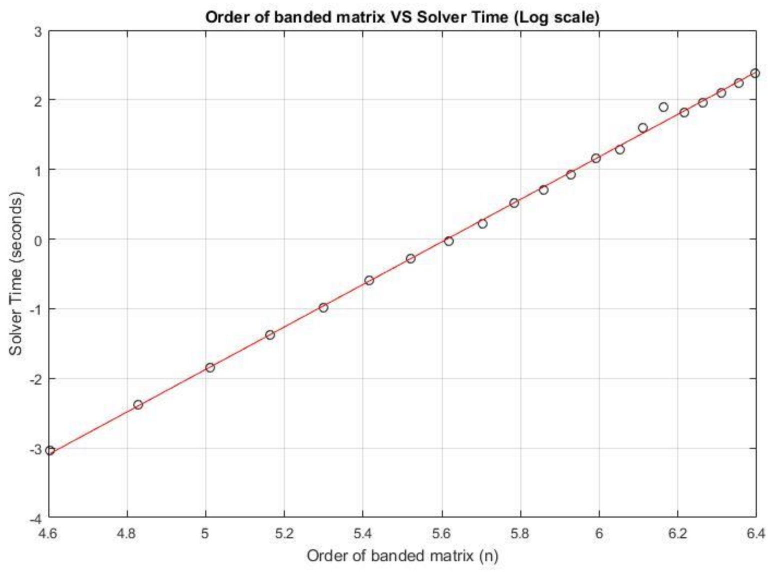
(b)

Fig 1: Solver time VS Order of random matrix (logarithmic scale) for (a) Code, (b) MATLAB operator

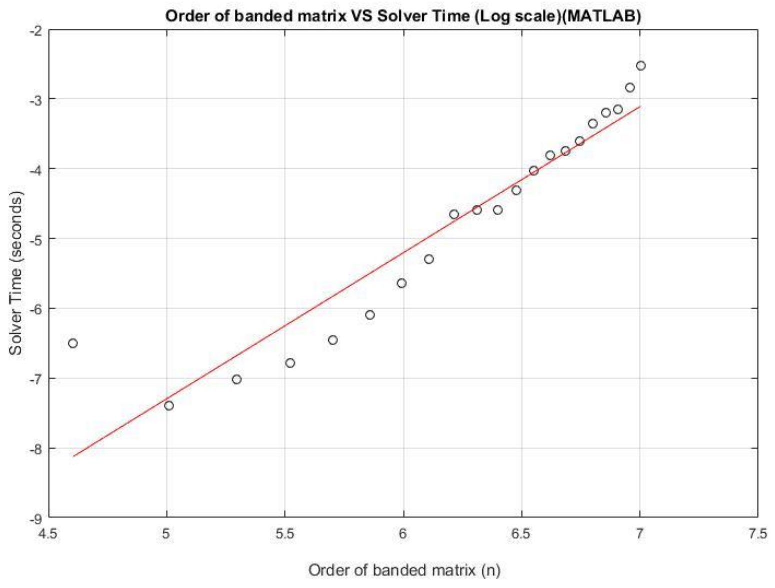
For different order of random matrices, the computational time is found and the corresponding graph is plotted. It is noticed that the best fit line for the Doolittle code scales as approximately n^3 (exact value 3.0019) whereas that for the MATLAB operator scales as approximately n^2 (exact value 2.0382).

3.2 LU Decomposition for Banded Matrix (Pivoting On)

The same experiment is conducted for a banded matrix with bandwidth equal to 5. The time taken by the algorithm to decompose the matrix using the code and the MATLAB operator is plotted against the order of the matrix and is represented in Figure 2.



(a)



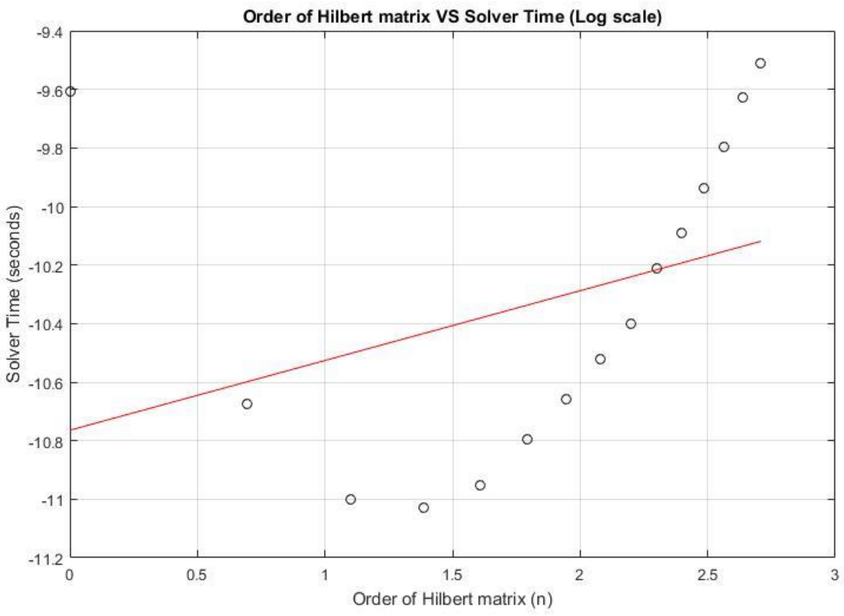
(b)

Fig. 2: Solver time VS Order of banded matrix (logarithmic scale) for (a) Code, (b) MATLAB operator

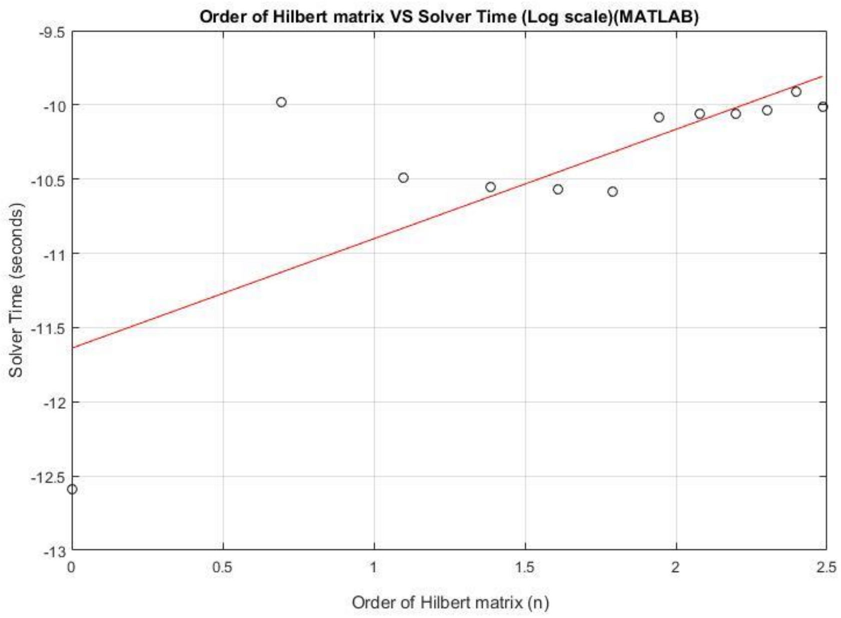
For different order of banded matrices, the computational time is found and the corresponding graph is plotted. It is noticed that the best fit line for the Doolittle code scales as approximately n^3 (exact value 3.0547) whereas that for the MATLAB operator scales as approximately n^2 (exact value 2.0932).

3.3 LU Decomposition for Hilbert Matrix (Pivoting On)

The experiment is then carried out for *n*-order Hilbert matrices. The plot of solver time versus the order of matrix is shown in Figure 3.



(a)



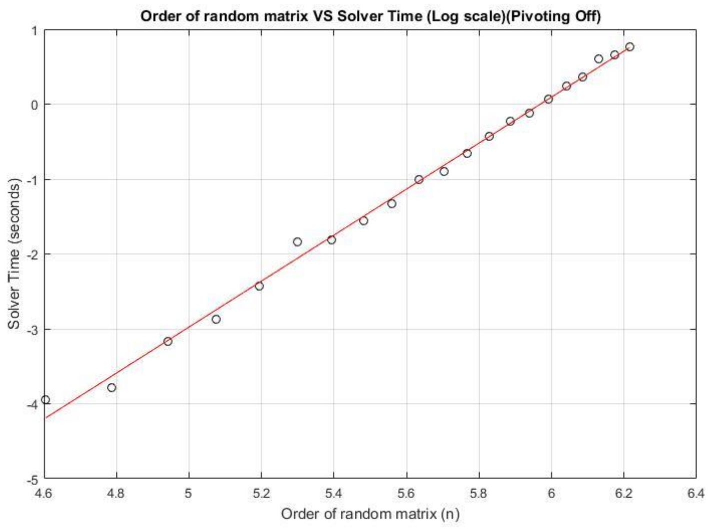
(b)

Fig. 3: Solver time VS Order of Hilbert matrix (logarithmic scale) for (a) Code, (b) MATLAB operator

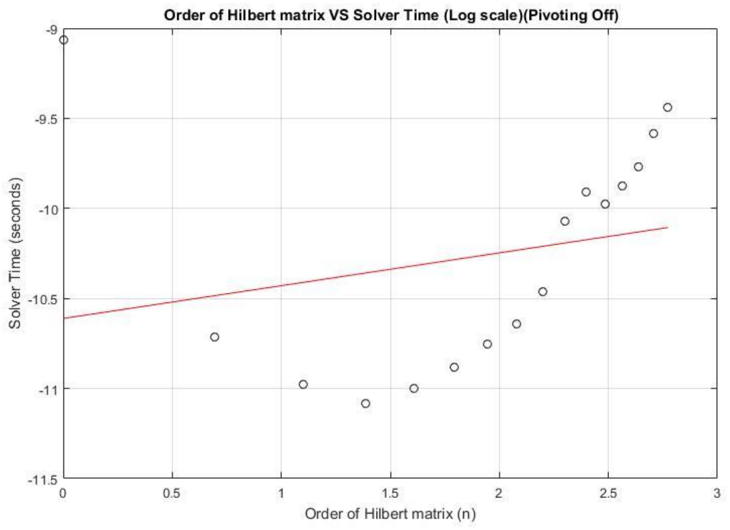
It is observed that the scaling for Hilbert matrices is arbitrary. Firstly, the LU decomposition using the Doolittle algorithm is achieved only up till a matrix of order 15. Further, using the MATLAB operator for LU decomposition, the decomposition is achieved only till an order 12. Beyond these orders, the matrix cannot be decomposed. Through these experiments, it can be observed that the Doolittle algorithm scales as n^3 whereas the MATLAB operator scales as n^2 for any matrix. However, since Hilbert matrices are ill-conditioned and mostly singular, it is impossible to decompose them beyond a certain order.

3.4 LU Decomposition for Random and Hilbert Matrices (Pivoting Off)

The algorithm is run turning the pivoting off to determine the performance and stability of the code. Tests are run for both random and Hilbert matrices to check whether the code is able to decompose the matrix. The scaling in both the cases is shown in Figure 4.



(a)



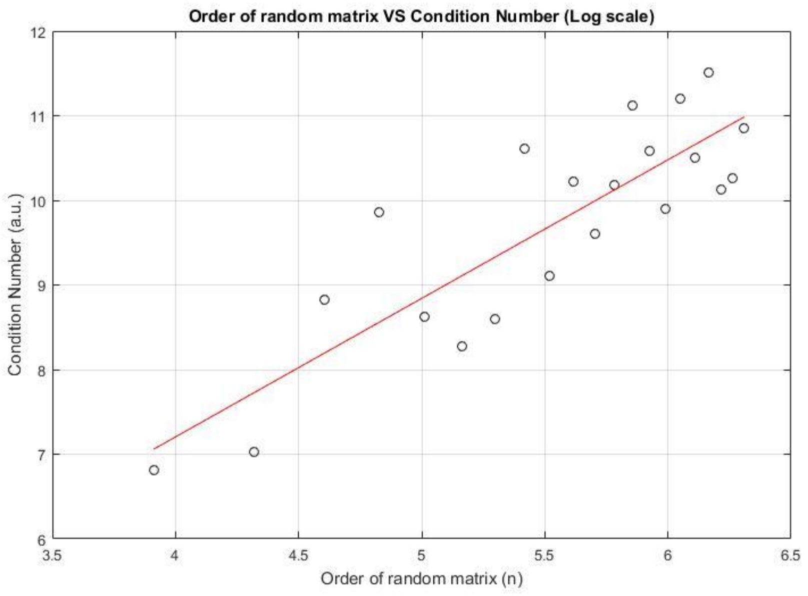
(b)

Fig. 4: Solver time VS Order of matrix (logarithmic scale)(Pivoting Off) for (a) Random, (b) Hilbert matrix

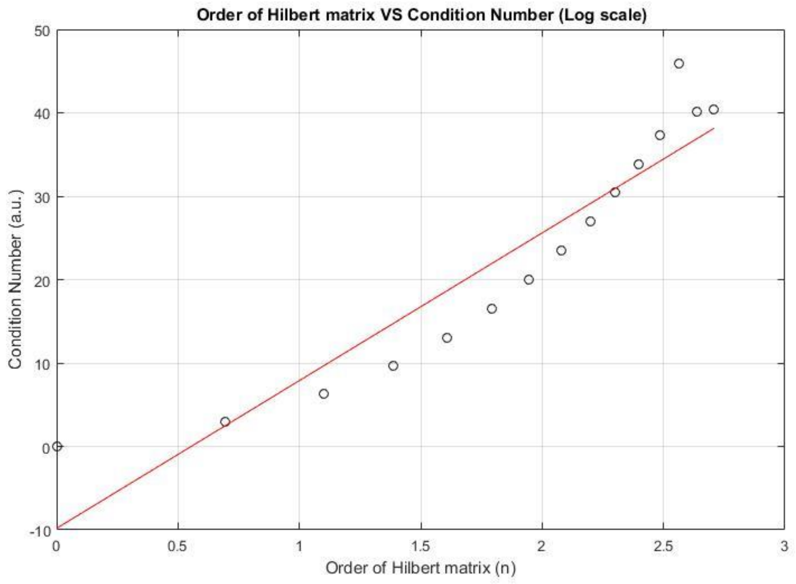
With the pivoting turned off, the random matrix still scales approximately as n^3 (exact value 3.0706) whereas the Hilbert matrix gives an arbitrary scaling (0.1821). With the pivoting turned off, the Hilbert matrices can be factored only till an order of 15. This can again be attributed to the fact that Hilbert matrices are ill-conditioned and thus, are not feasible in numerical computation. The random matrix, however, can be still be factored with pivoting turned off.

3.5 Condition number versus Order of matrix for Random and Hilbert Matrices

The variation in the condition number with the order of the matrix for both the random and the Hilbert matrices is tested and the results of the same are shown in Figure 5.



(a)



(b)

Fig. 5: Condition Number VS Order of matrix (logarithmic scale) for (a) Random, (b) Hilbert matrix

On plotting the condition number for both cases, it is observed that the condition number does not seem to depend on the order of the matrix in case of random matrices. However, for Hilbert matrices, the condition number increases as the order increases. The condition number determines the ability of the algorithm to find an approximate solution to the system of equations. As the condition number nears 1, greater will be the accuracy of the approximate solution. However, as condition number goes higher, the problem does not possess a unique, well-defined solution. This also means that the matrix is not invertible which holds true in case of Hilbert matrices which are largely singular. In such cases, the matrix is said to be ill-conditioned and the solution of a linear system of equations is prone to large numerical errors.

3.6 LU Decomposition for System of Equations with Varying Powers

In this particular experiment, a 20 $\times$20 random matrix, *A*, is decomposed for a random matrix *B* of the order 20 $\times$1. Hence, the equation that is solved is exactly the same as Equation (1). For the same matrices *A* and *B*, the system of equations for varying powers, as in Equation 5, are solved to analyze the behavior of the solver and the solution.

$$A^n x = B \tag{5}$$

It is observed that for a 20 $\times$20 matrix decomposition, as the value of *n* increases, the matrix cannot be decomposed. For every square matrix representing a large system of equations, the matrix does not decompose for higher values of the power.

4. Conclusion

By running different test cases for different matrix types, we can conclude that the Doolittle algorithm scales as n^3. The MATLAB intrinsic operator for the same LU decomposition scales as n^2. This means that the code developed using the Doolittle algorithm is much slower and less efficient as compared to the MATLAB intrinsic operator.

Further, it is seen that on turning pivoting off, the scaling of the algorithm is still n^3. This however, is not the same for Hilbert matrices. On applying the decomposition algorithm to Hilbert matrices, the scaling is arbitrary value which is neither n^2 nor n^3. Moreover, turning pivoting on or off causes Hilbert matrices to be decomposed only to a certain order, in this case the order being 15.

Comparing the condition number with the order of the matrix tells us how accurate the approximation to the solution of the problem might be. In case of singular matrices, the condition number reaches abnormally high values as is seen in the case of Hilbert matrices. The condition number increases with increase in order of Hilbert matrices. However, the trend is not similar in case of random matrices and it can be said that the condition number in case of random matrices is independent of the order.

Solving a system of equations with varying power tells us that on increasing the power, the tendency of the matrix to get decomposed decreases. As the power increases, the matrix is unable to decompose using the Doolittle algorithm.

All the tests conducted as part of this project give us a deeper insight into the workings of the Doolittle algorithm and a clear idea on what scaling actually is. For different types of matrices, the behavior of the algorithm is studied, thus, creating a clear picture to what factors govern the ability to solve a system of linear equations.

5. Appendix

The MATLAB code for the LU decomposition using the Doolittle algorithm is provided as follows.

```
% LU Decomposition

function [x,t] = ludecomp(a,b,n,tol)

    o = zeros(n,1);                        % Sets vector o = 3x1
    s = zeros(n,1);                        % Sets vector s = 3x1
    er = 0;
    %tic;                                  % Start timer
    [a,o] = decompose(a,n,tol,o,s);        % Calls function decomposition
    if er ~= -1
        [a,o,x] = substitute(a,o,n,b);     % Calls function substitution
    end
    toc;
    disp(x)                                % Displays solution
end

% Decomposition

function [a,o] = decompose(a,n,tol,o,s)
    for i = 1:n
        o(i) = i;
        s(i) = abs(a(i,i));
        for j = 2:n
            if abs(a(i,j)) > s(i)
                s(i) = abs(a(i,j));
            end
        end
    end
    for k = 1:n-1
        o = pivot(a,o,s,n,k);              % Calls function pivot
        if abs(a(o(k),k)/s(o(k))) < tol
            er = -1;
            disp (a(o(k),k)/s(o(k)));
            break
        end
        for i = k+1:n
            factor = a(o(i),k)/a(o(k),k);
            a(o(i),k) = factor;
            for j = k+1:n
                a(o(i),j) = a(o(i),j) - factor*a(o(k),j);
            end
        end
    end
    if abs(a(o(k),k)/s(o(k))) < tol
        er = -1;
        disp (a(o(k),k)/s(o(k)));
    end
end
```

```
% Pivot

function o = pivot(a,o,s,n,k)
    p=k;
    big = abs(a(o(k),k)/s(o(k)));
    for ii = k+1:n
        dummy = abs(a(o(ii),k)/s(o(ii)));
        if dummy > big
            big = dummy;
            p = ii;
        end
    end
    dummy = o(p);
    o(p) = o(k);
    o(k) = dummy;
end

% Substitution

function [a,o,x] = substitute(a,o,n,b)
    for i = 2:n
        sum = b(o(i));
        for j = 1:i-1
            sum = sum - a(o(i),j)*b(o(j));
        end
        b(o(i)) = sum;
    end

    x = zeros(n,1);
    x(n) = b(o(n))/a(o(n),n);
    for i = n-1:-1:1
        sum = 0;
        for j = i+1:n
            sum = sum + a(o(i),j)*x(j);
        end
        x(i) = (b(o(i))-sum)/a(o(i),i);
    end
end

% Generate Hilbert Matrix

function a = Hilbert(n)
    for i = 1:n
        for j = 1:n
            a(i,j)=1/(i+j-1);
        end
    end
end
```

```
% Generate Banded Matrix

function a = banded(n,w)
    a = zeros(n);
    for i = 1:n
        for j = i-w:i+w
            if j>0
                if j<n+1
                    a(i,j)=rand;
                end
            end
        end
    end
end

% Code for Scaling

function scaling(k,tol)
    n = zeros(k+1,1);
    m = zeros(k+1,1);
    l = zeros(k+1,1);
    n(1) = 20;
    for i = 1:k+1
        [x,t] = ludecomp(n(i),tol);
        n(i+1)=n(i)+20;
        l(i) = log(n(i));
        m(i) = log(t);
        plot(l(i),m(i),'ok');
        hold on;
    end
    q = polyfit(l,m,1)
    y = polyval(q,l);
    plot(l,y,'-r')
    title('Order of random matrix VS Solver Time (Log scale)');
    xlabel('Order of random matrix (n)');
    ylabel('Solver Time (seconds)');
    grid on;
end

% MATLAB intrinsic LU Decomposition

function [x,t] = matlabLUdecomp(n)
    a=rand(n);
    b=rand(n,1);
    tic
    x=a\b;
    t = toc;
end
```