09 April 2019

Warsaw University of Technology – EiTI faculty

Numerical methods for solving systems of linear algebraic equations

No. 38

Wiktor Bara

Course name: Numerical Methods

Advisor: Andrzej Miękina

Table of contents

Introduction

The numerical errors and uncertainties that occur during different numerical operations can oscillate in various ways and can differ to each other especially taking under considerations the system of equations. The operations and dependence of those operations included in this report are determinant (**det**) and condition number of inversion (**cond**), that are taking under consideration below matrix $\mathbf{A}_N(x)$, are described in the first section. While the LU factorization, LLT factorization (Choleski decomposition), inversion based on both algorithms and the norms checking the accuracy of those are described further.

$$\mathbf{A}_N(x) = \begin{bmatrix} x^2 & -\frac{2x}{3} & -\frac{2x}{3} & -\frac{2x}{3} & \cdots & -\frac{2x}{3} & -\frac{2x}{3} \\ -\frac{2x}{3} & \frac{8}{9} & \frac{8}{9} & \frac{8}{9} & \cdots & \frac{8}{9} & \frac{8}{9} \\ -\frac{2x}{3} & \frac{8}{9} & \frac{12}{9} & \frac{12}{9} & \cdots & \frac{12}{9} & \frac{12}{9} \\ -\frac{2x}{3} & \frac{8}{9} & \frac{12}{9} & \frac{16}{9} & \cdots & \frac{16}{9} & \frac{16}{9} \\ -\frac{2x}{3} & \frac{8}{9} & \frac{12}{9} & \frac{16}{9} & \cdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \frac{(N-1)\cdot 4}{9} & \frac{(N-1)\cdot 4}{9} \\ -\frac{2x}{3} & \frac{8}{9} & \frac{12}{9} & \frac{16}{9} & \cdots & \frac{(N-1)\cdot 4}{9} & \frac{N\cdot 4}{9} \end{bmatrix}$$

Figure 1 Scheme of considered matrix

Description of Numerical Algorithms

The Gauss method of elimination (LU factorization)

This algorithm creates form given matrix two different matrices, L(lower triangle) and U(upper triangle), multiplication of which is equal to initial matrix. The algorithm aims to obtain two matrices that in corresponding triangles are non zero values while the rest of matrix is zero. The elements on diagonal in L matrix, are in assumption equal 1. The example of created matrices are in Figure 2.

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 6 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{3} & 2 & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 3 & 1 & 6 \\ 0 & \frac{1}{3} & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2. Example of matrices in LU decomposition

$$\boldsymbol{L \cdot U = A}$$

(1.1)

Cholesky-Banachiewicz factorization (LLT factorization)

The algorithm creates from given matrix two matrices, where one is transposition of other one, multiplication of which is equal to initial matrix. Moreover, the matrices are divided into the lower and upper one. In each of them the elements in corresponding triangle are non zero and the rest of them is equal to zero. The example of general matrices obtained with this algorithm are on Figure 3.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{bmatrix} = \begin{bmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{N,1} & l_{N,2} & \cdots & l_{N,N} \end{bmatrix} \cdot \begin{bmatrix} l_{1,1} & l_{2,1} & \cdots & l_{N,1} \\ 0 & l_{2,2} & \cdots & l_{N,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{N,N} \end{bmatrix}$$

Figure 3. Example of matrices in LLT decomposition

$$\boldsymbol{L \cdot L^T = A}$$

(1.2)

The value of particular element is given by the equations below.

$$l_{n,n} = \sqrt{a_{n,n} - \sum_{i=1}^{n-1} l_{n,i}^2}$$

(1.3)

$$l_{v,n} = \frac{a_{v,n} - \sum_{i=1}^{n-1} l_{v,i} \cdot l_{n,i}}{l_{n,n}}$$

(1.4)

for n = 1,…,N , for v = n+1,…,N where N is size of Matrix

Methodology

The consideration of determinant of **A**$_N$(x) when the x = sin(α – 1).

The determinant is an algebraic sum of product from all elements of matrix. Each of those products includes one number from first row and one number from first column. Setting all values in first row or column implies that determinant of matrix will be equal zero. From this we can derive that:

$$\det(\boldsymbol{A}) = 0 <=> x = 0$$

(2.1)

It implies that the smallest value of $\alpha = \alpha_N = 1$.

The dependence of det(**A**) and cond(**A**) for $\alpha = [\alpha_N – 0.01, \alpha_N + 0.01]$ are presented on the Fig.4 and Fig.5

The procedure of inverting matrix

The procedure of inverting matrix **A**$_\mathbf{N}$ is presented in the equation below:

$$\boldsymbol{A} \cdot \boldsymbol{A}^{-1} = \boldsymbol{I}$$

(2.2)

where **I** is the identity matrix with dimensions N x N.

Denoting $\mathbf{A}^{-1}$ and **I** columns as respectively $\mathbf{y}_N$ and $\mathbf{e}_N$, where the N is the number of column, the equation above will looks like:

$$\boldsymbol{A} \cdot \boldsymbol{y}_N = \boldsymbol{e}_N$$

(2.3)

After substitution LU factorization:

$$\boldsymbol{U} \cdot \boldsymbol{L} \cdot \boldsymbol{y}_N = \boldsymbol{e}_N$$

(2.4)

After substitution of LLT factorization:

$$\boldsymbol{L} \cdot \boldsymbol{L}^{\boldsymbol{T}} \cdot \boldsymbol{y}_N = \boldsymbol{e}_N$$

(2.5)

The correctness and accuracy of those methods are tested with MATLAB operation inv.

To compare the results from those methods the following norms were calculated

$$\delta_2 = \left\| \boldsymbol{A}_{N,x} \cdot \boldsymbol{A}_{N,x}^{-1} - \boldsymbol{I}_N \right\|_2 \quad \text{(the root-mean-square error) RMSE}$$

(2.6)

$$\delta_\infty = \left\| \boldsymbol{A}_{N,x} \cdot \boldsymbol{A}_{N,x}^{-1} - \boldsymbol{I}_N \right\|_\infty \quad \text{(the maximum error) ME}$$

(2.7)

where the matrices $\boldsymbol{A}_{N,x}, \boldsymbol{A}_{N,x}^{-1}$ with $N \in \{3,10,20\}$ and $x = \frac{2^k}{300}, k \in \{0,1,2,\dots,21\}$.

Results

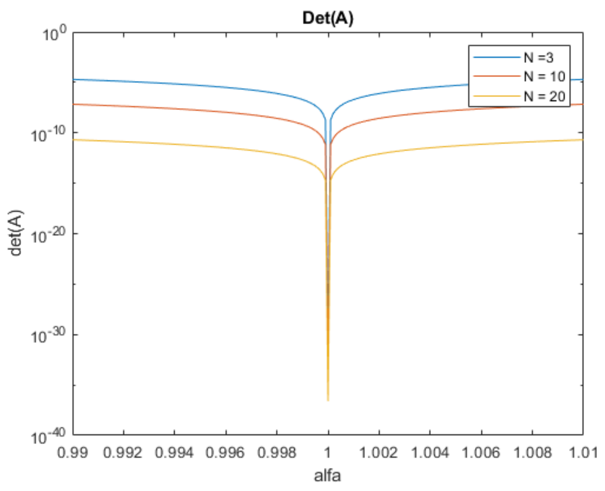

Figure 5. Dependence of determinant of Matrix $A_{N,x}$ on values of alfa

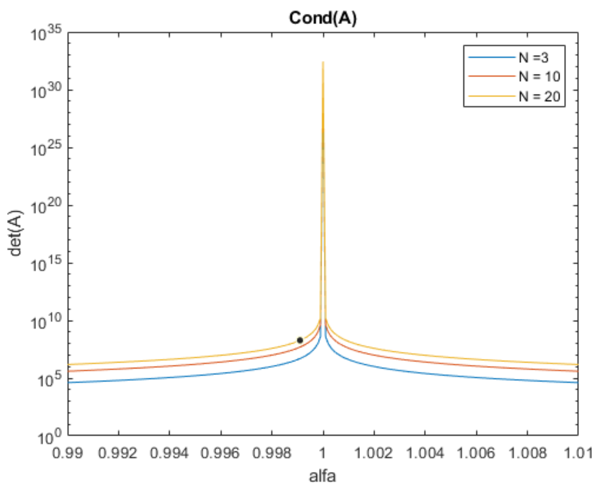

Figure 4. Dependence of condition number of inversion of Matrix $A_{N,x}$ on values of alfa

In the Fig.4 the determinant tends to 0 when the value of alfa tends to 1. Due to the lack of approximation in the decimal display of x=(1-α) for x tending to 0, the difference in the determinant of the corresponding values of alfa is rather small considering the size of matrix. For bigger matrices the bigger amount of confirmed zeros in the first row and column provide a little more accurate values. In the condition number of inversion the figure (Fig.5) is analogous to the determinant figure, the only difference is that the value of this tends to infinity when alfa tends to 1 and the largest value of cond is for largest matrix.

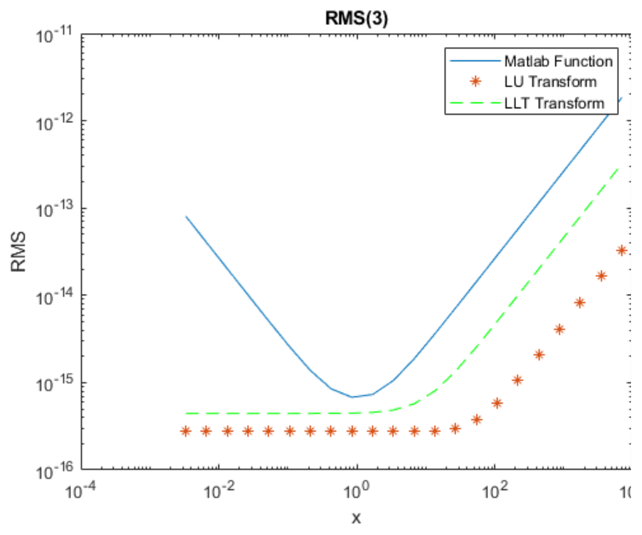

Figure 7. Comparison RMSE of different inverse algorithms for matrix 3x3

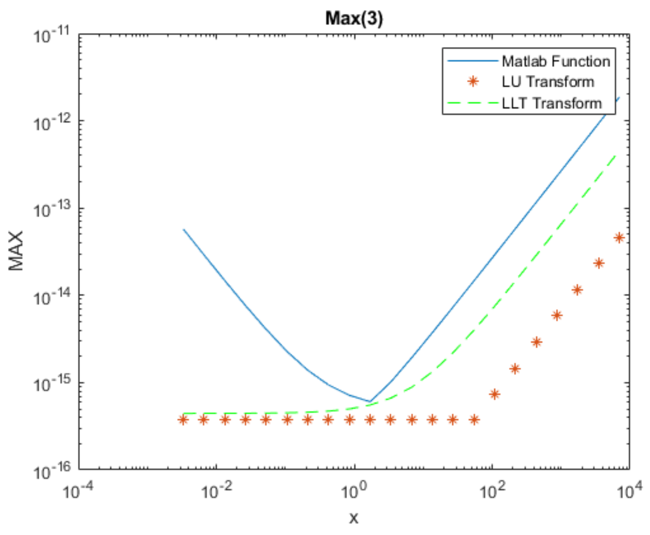

Figure 6. Comparison ME of different inverse algorithms for matrix 3x3

The difference of both norms under consideration different inverse algorithms shows that for small Matrix 3x3 the less accurate algorithm is the built in one. From LU and LLT transform the best is LU.

wThe difference in figures Fig.6 , Fig.7 is almost neglectable. The Max Error one is a bit sharper and the LU and LLT are a little bit closer to each other at small values of x.

The smallest value of norm is in the small interval of x from the smallest value to almost x=100. Operations carried on matrix of this size are the most accurate for LU transform.

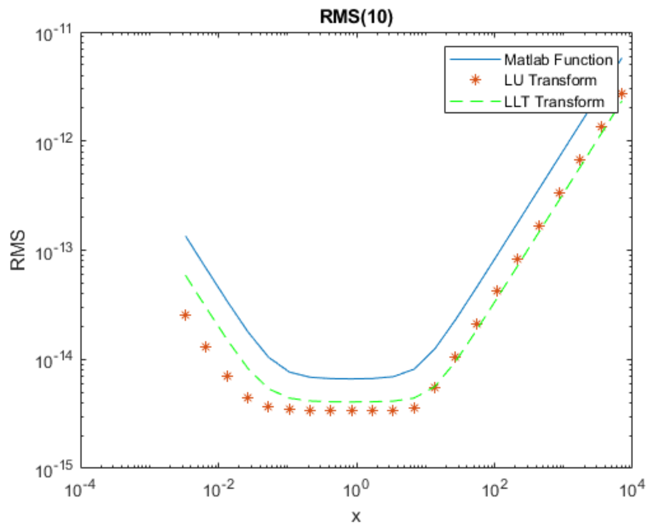

Figure 9. Comparison RMSE of different inverse algorithms for matrix 10x10

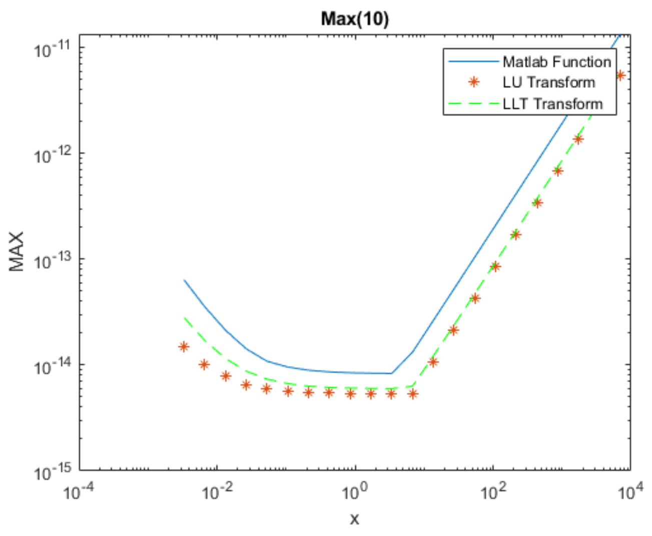

Figure 8. Comparison ME of different inverse algorithms for matrix 10x10

For the bigger matrices (10x10) the differences in both LU (Fig.8) and LLT (Fig.9) transform cover up and the norms are almost the same.

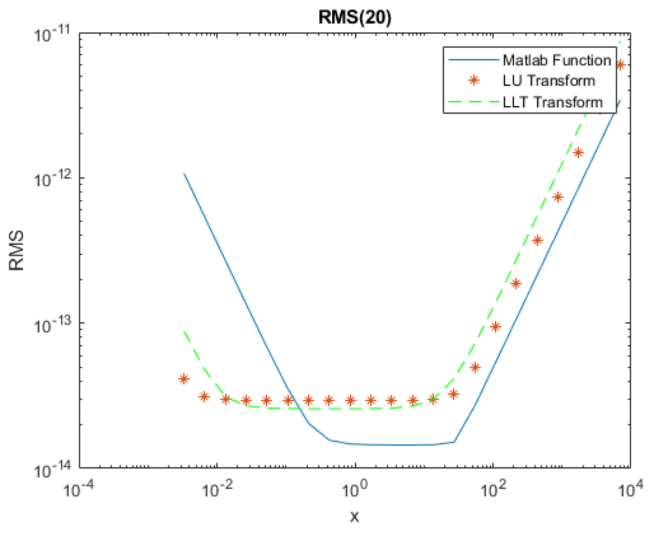

Figure 10. Comparison RMSE of different inverse algorithms for matrix 20x20

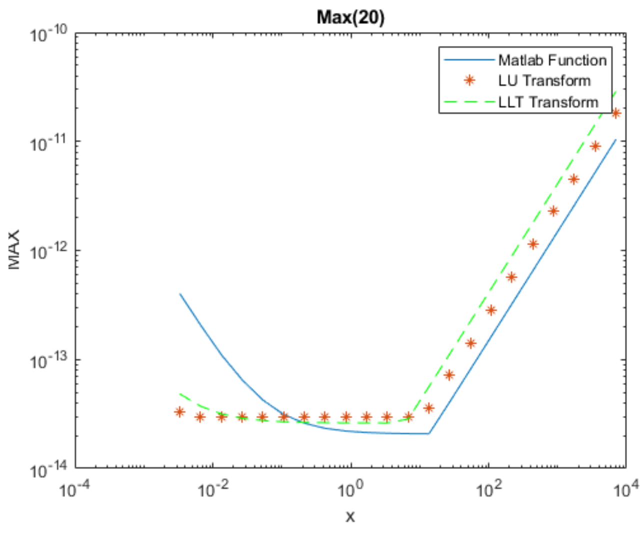

Figure 11. Comparison ME of different inverse algorithms for matrix 20x20

In case of 20x20 matrices the differences are almost none but the built in function is more accurate for x bigger than 0.l.

But in case of using matrices that not exceed 10 that superior algorithm to compute the inverse matrix of that type is the LU transform. For matrices that exceed 20 is sure that for x that are bigger than 0.1 the preferable algorithm is the built in one and the accuracy is a little better in those cases.

Conclusions

There exists the dependence between determinant and condition number of inversion of given matrix. The determinant is ,inversely proportional to condition number of inversion, for $\alpha_N = 1$ each element of first row and first column of matrix is equal zero and that implies that determinant is equal zero in that point on the other hand the condition number due to its dependence on determinant tends to infinity.

The inversion of matrix and it accuracy depends on the algorithm used to inverse. For smaller matrices the best option is the LU transform. It is a bit better than LLT transform and the difference to built in function is significant. For bigger matrices the built in function turned out to be the best for bigger values of x. For very small values of x the LU transform still is better that any other algorithm.

List of References

[1] https://www.mathworks.com,

[2] Lecture notes ENUME 2019, Roman Morawski

Source Code

```
clear all
close all
[alfa3, alfaValues3, detValues3, condValues3] =
minAlfa(10e-5, 10e-5, 3);
[alfa10, alfaValues10, detValues10, condValues10]
= minAlfa(10e-5, 10e-5, 10);
[alfa20, alfaValues20, detValues20, condValues20]
= minAlfa(10e-5, 10e-5, 20);


%-----


figure('Name', 'Det(A)')
set(gca, 'YScale', 'log')
plot(alfaValues3, detValues3)
hold on

set(gca, 'YScale', 'log')
plot(alfaValues10, detValues10)
hold on

set(gca, 'YScale', 'log')
plot(alfaValues20, detValues20)
hold on
title('Det(A)')
xlabel('alfa')
ylabel('det(A)')
legend('N =3', 'N = 10', 'N = 20')

hold off
%------

figure('Name', 'Cond(A)')
set(gca, 'YScale', 'log')
plot(alfaValues3, condValues3)
hold on

set(gca, 'YScale', 'log')
plot(alfaValues10, condValues10)
hold on

set(gca, 'YScale', 'log')
plot(alfaValues20, condValues20)
hold on
title('Cond(A)')
xlabel('alfa')
ylabel('det(A)')
legend('N =3', 'N = 10', 'N = 20')

hold off
%---------------------------------

cnt=0;
for n=[3,10,20]
    cnt=cnt+1;
    A{1,cnt}=createMatrix(n,1/300);
    B{1,cnt}=invUL(A{1,cnt});
    C{1,cnt}=invLLT(A{1,cnt});
    I=eye(n);
    ARMS(1,cnt) = RMSEr(A{1,cnt}*inv(A{1,cnt}) -
I);
    BRMS(1,cnt) = RMSEr(A{1,cnt}*B{1,cnt} - I);
    CRMS(1,cnt) = RMSEr(A{1,cnt}*C{1,cnt} - I);
    AMax(1,cnt) = MaxEr(A{1,cnt}*inv(A{1,cnt}) -
I);
    BMax(1,cnt) = MaxEr(A{1,cnt}*B{1,cnt} - I);
    CMax(1,cnt) = MaxEr(A{1,cnt}*C{1,cnt} - I);

    for k=2:22
        A{k,cnt} = A{k-1,cnt};
        A{k,cnt}(:,1) = A{k,cnt}(:,1)*2;
        A{k,cnt}(1,:) = A{k,cnt}(1,:)*2;
        B{k,cnt}=invUL(A{k,cnt});
        C{k,cnt}=invLLT(A{k,cnt});
        ARMS(k,cnt) =
RMSEr(A{k,cnt}*inv(A{k,cnt}) - I);
        BRMS(k,cnt) = RMSEr(A{k,cnt}*B{k,cnt} -
I);
        CRMS(k,cnt) = RMSEr(A{k,cnt}*C{k,cnt} -
I);
        AMax(k,cnt) =
MaxEr(A{k,cnt}*inv(A{k,cnt}) - I);
        BMax(k,cnt) = MaxEr(A{k,cnt}*B{k,cnt} -
I);
        CMax(k,cnt) = MaxEr(A{k,cnt}*C{k,cnt} -
I);
    end
end

for k=0:21
    x(k+1) = 2^k/300;
end

%-------
cnt = 1;
for k=[3,10,20]
    figure('Name', ['RMS(',num2str(k),')'])
    loglog(x,ARMS(:,cnt))
    hold on

    loglog(x,BRMS(:,cnt),'*')
    hold on

    loglog(x,CRMS(:,cnt),'g--')
    hold on
    title(['RMS(',num2str(k),')'])
    xlabel('x')
    ylabel('RMS')

    hold off
    legend('Matlab Function', 'LU Transform',
'LLT Transform')

    figure('Name', ['Max(',num2str(k),')'])
    loglog(x,AMax(:,cnt))
    hold on

    loglog(x,BMax(:,cnt),'*')
    hold on

    loglog(x,CMax(:,cnt),'g--')
    hold on
    title(['Max(',num2str(k),')'])
    xlabel('x')
    ylabel('MAX')

    hold off
    legend('Matlab Function', 'LU Transform',
'LLT Transform')

    cnt=cnt+1;
end
```

```
function er = RMSEr(A)
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
er = max(sqrt(abs(eig(transpose(A)*A))));
end

function er = MaxEr(A)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
er = max(sum(abs(A),2));
end

function [U,L] = LU(A)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
U = A;
n = size(A,1);
L = eye(n);
    for i=1:n-1
        for j=i+1:n
                L(j,i)=U(j,i)/U(i,i);
                U(j,:) = U(j,:)-L(j,i)*U(i,:);
        end
    end

end

function [T,L] = LLT(A)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
n = size(A,1);
L = zeros(n);
    for i=1:n
        L(i,i)=sqrt(A(i,i) - sum(L(i,:).^2));
        for j=(i+1):n
            L(j,i) = (A(j,i) -
sum(L(i,:).*L(j,:)))/L(i,i);

        end
    end
T = transpose(L);

end

function [A] = invUL(B)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
[U,L] = LU(B);
n = size(U,1);
I = eye(n);
    for i=1:n
        X(1,i) = I(1,i);
        for j=2:n
            X(j,i) = (I(j,i)-L(j,1:j-1)*X(1:j-
1,i));
        end
        A(n,i) = X(n,i)./U(n,n);
        for j=(n-1):-1:1
            A(j,i) = (X(j,i)-
U(j,j+1:n)*A(j+1:n,i))./U(j,j);
        end
    end
end

function [A] = invLLT(B)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
[T,L] = LLT(B);
n = size(T,1);
I = eye(n);
    for i=1:n
        X(1,i) = I(1,i)./L(1,1);
        for j=2:n
            X(j,i) = (I(j,i)-L(j,1:j-1)*X(1:j-
1,i))./L(j,j);
        end
        A(n,i) = X(n,i)/T(n,n);
        for j=(n-1):-1:1
            A(j,i) = (X(j,i)-
T(j,j+1:n)*A(j+1:n,i))./T(j,j);
        end
    end
end

function [alfa, alfaValues, detValues,
condValues] = minAlfa(precision,difference,n)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
alfa = 1;
    %while (det(createMatrix(n,sin(alfa - 1))) >
precision)
     %   alfa = alfa + difference;
    %end
    %while(det(createMatrix(n,sin(alfa - 1))) >
det(createMatrix(n,sin(alfa - 1 + difference))))
     %  alfa = alfa + difference;
    %end

    temp = alfa-0.01;
    i=1;
    while temp < alfa+0.01
        alfaValues(i) = temp;
        detValues(i) =
det(createMatrix(n,sin(temp-1)));
        condValues(i) =
cond(createMatrix(n,sin(temp-1)));
        i = i+1;
        temp = temp + difference;
    end
end

function A = createMatrix(n,x)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

    for k=n:-1:2
        A(k,k:n)=(k*4)/9;
        A(k:n,k)=(k*4)/9;
    end

    A(1,2:n)=(-2*x)/3;
    A(2:n,1)=(-2*x)/3;
    A(1,1)=x^2;
end
```