22 April 2020

Emilia Wróblewska
Group 4

Computing $X = A^{-1}$ using Crout's method, where A is symmetric and positive definite. Finding LU-decomposition of matrix A, computing U^{-1} and solving system of linear equations $XL = U^{-1}$.

1 Method description

In numerical analysis and linear algebra, LU decomposition factors a matrix as the product of a lower triangular matrix and an upper triangular matrix.

An LU factorization refers to the factorization of matrix A, with proper row and/or column orderings or permutations, into two factors, a lower triangular matrix L and an upper triangular matrix U:

$$A = LU$$

Crout's method provides a way to factor matrix A into an LU decomposition without using Gaussian Elimination.

For a general $n \times n$ matrix A, we assume that LU decomposition exists, and write the form of L and U explicitly. The factorization can be computed directly from $A = LU$, so we set $U = I_n$ to be the identity matrix and L to be $n \times n$ zero matrix. We then compute:

for $k = 1, ..., n$

$$l_{i,k} = a_{i,k} - \sum_{p=0}^{k-1}(l_{i,p} \cdot u_{p,k})$$

for $k = k, k+1, ..., n$ produces the k$^{\text{th}}$ row of L.

$$u_{k,j} = \frac{a_{k,j} - \sum_{p=1}^{k-1}(l_{k,p} \cdot u_{p,j})}{l_{k,k}}$$

for $j = k+1, k+2, , ..., n$ produces the k$^{\text{th}}$ row of U.

Thanks to this we can calculate A^{-1} using the identity:

$$A^{-1} = (LU)^{-1} = U^{-1} \cdot L^{-1}$$

This calculations provide us the LU-factorization of A and U^{-1} already computed, therefore we just need to solve for X the equation:

$$X \cdot L = U^{-1}$$

However, we can easily observe that calculating X requires just moving the matrix L to the right side of the equation. And thus, applying the identity, we obtain:

$$X = U^{-1} \cdot L^{-1} = A^{-1}$$

which ends our calculations, because the result is $X = A^{-1}$.

2 Program description

After You ran the program, You will see the Menu:

Menu

Input matrix A

Display variable

Compute inverse of A

Find LU-factorization of A

Compute inverse of U

Compute $X \cdot L = U^{-1}$

Run tests

FINISH

- If You press button „Input matrix A", You will be able to input Your own matrix for calculations. The program will immediately check if Your matrix is correct i.e is symmetric and positive definite. If not - the program will keep asking You to input another matrix until it meets all conditions required. If You do not enter anything - the default argument will be used.
- Option „Display variable" will simply display previously input variable.
- After clicking „Compute inverse of A" the matrix inverse will be calculated using Crout's method of finding LU-decomposition.
- Option „Find LU-factorization of A" will find LU-decomposition of A by Crout's algorithm and display obtained matrices L and U.
- „Compute inverse of U" will compute inverse of upper triangular matrix U in LU-factorization of A.
- Option „Compute $X \cdot L = U^{-1}$" will solve the system of linear equations for found L and U^{-1} and display the resulting matrix X.

- Option „Run tests" will start the function $CroutErrors$ which calculates errors occuring while using crout_inverse algorithm, Crout algorithm and solving the equation $XL = U^{-1}$.
- At last „FINISH" will end the program.

MATLAB functions used:

1. Menu_Crout.m - script for graphic interface for the rest of the functions.
2. Crout.m - function strictly for computing LU-decomposition from given matrix A using formulas described at the beginning.
3. crout_inverse.m - function calculating 3 types of errors for one of 3 selected example.
4. invlower.m & invupp.m - functions from laboratories used for computing inverse matrices for lower and upper triangular matrices respectively.
5. check_matrix.m - function checking if input matrix is symmetric and positive definite.
6. CroutErrors.m - function calculating and displaying errors of algorithms used in the program.

   (**Note.** *All source codes can be found in section 5.*)

3 Numerical tests

All the numerical tests are included in the CroutErrors.m file. It works by implementing a simple menu in which the user can choose some specified example. According to user's choice the script sets appropriate A as either the Pascal or Hilbert matrix of fixed size, which guarantees that A will be symmetric and positive definite and thus meet the necessary assumptions. Firstly, the program calls $crout\_inverse$ function to compare its result with Matlab built-in function $inv()$ and calculates errors of this solution. Then the program calls $Crout$ algorithm and calculates the LU-decomposition error. And at last, errors of solving the system $X \cdot L = U^{-1}$ are computed.

1. Errors occuring while computing the inverse of A, where I - the Identity matrix and X - the inverse of A calculated using $crout\_inverse$.
   (a) Right residual error
   $$\frac{||AX - I||}{||A|| \cdot ||X||}$$
   (b) Left residual error
   $$\frac{||XA - I||}{||A|| \cdot ||X||}$$

(c) Final error

$$\frac{||X^{-1}A - A||}{||A||}$$

2. The LU-decomposition error

   (a) Error of lower triangular matrix

$$\frac{||A - L \cdot L^T||}{||A||}$$

   (b) Error of upper triangular matrix

$$\frac{||A - U \cdot U^T||}{||A||}$$

3. Errors of solving the system $XL = U^{-1}$, where Z is the exact solution of our problem, namely A^{-1}.

   (a) Relative error

$$\frac{||X - Z||}{||Z||}$$

   (b) Forward stability error

$$\frac{||X - Z||}{||Z||cond(A)}, \quad \text{where } cond(A) = ||A^{-1}|| \cdot ||A||$$

   (c) Backward stability error

$$\frac{||U^{-1} - AX||}{||A|| \cdot ||X||}$$

Tests: X is the inverse of our initial matrix A obtained using our implemented $crout\_inverse$ function, whereas M is the matrix calculated using predefined Matlab function $inv()$.

1. $A = pascal(3)$ and $X = A^{-1}$

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 6 \end{pmatrix}$$

$$X = \begin{pmatrix} 3 & -3 & 1 \\ -3 & 5 & -2 \\ 1 & -2 & 1 \end{pmatrix}, \; M = \begin{pmatrix} 3.0000 & -3.0000 & 1.0000 \\ -3.0000 & 5.0000 & -2.0000 \\ 1.0000 & -2.0000 & 1.0000 \end{pmatrix}$$

Results:

Tablica 1: Inverse errors

Right residual error	Left residual error	Total error
0	0	$4.2899 \cdot 10^{-16}$

Tablica 2: LU-decomposition errors

Lower error	Upper error
0	0.6519

Tablica 3: $cond(A)$ & Equation errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
61.9839	0	0	0.0369

2. $A = pascal(5)$ and $X = A^{-1}$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{pmatrix}$$

$$X = \begin{pmatrix} 5 & -10 & 10 & -5 & 1 \\ -10 & 30 & -35 & 19 & -4 \\ 10 & -35 & 46 & -27 & 6 \\ -5 & 19 & -27 & 17 & -4 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 5.0000 & -10.0000 & 10.0000 & -5.0000 & 1.0000 \\ -10.0000 & 30.0000 & -35.0000 & 19.0000 & -4.0000 \\ 10.0000 & -35.0000 & 46.0000 & -27.0000 & 6.0000 \\ -5.0000 & 19.0000 & -27.0000 & 17.0000 & -4.0000 \\ 1.0000 & -4.0000 & 6.0000 & -4.0000 & 1.0000 \end{pmatrix}$$

Results:

Tablica 4: Inverse errors

Right residual error	Left residual error	Total error
0	0	$4.1952 \cdot 10^{-15}$

Tablica 5: LU-decomposition errors

Lower error	Upper error
0	0.8841

Tablica 6: $cond(A)$ & Equation errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$8.5175 \cdot 10^3$	0	0	0.0011

3. $A = pascal(10)$ and $X = A^{-1}$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 3 & 6 & 10 & 15 & 21 & 28 & 36 & 45 & 55 \\ 1 & 4 & 10 & 20 & 35 & 56 & 84 & 120 & 165 & 220 \\ 1 & 5 & 15 & 35 & 70 & 126 & 210 & 330 & 495 & 715 \\ 1 & 6 & 21 & 56 & 126 & 252 & 462 & 792 & 1287 & 2002 \\ 1 & 7 & 28 & 84 & 210 & 462 & 924 & 1716 & 3003 & 5005 \\ 1 & 8 & 36 & 120 & 330 & 792 & 1716 & 3432 & 6435 & 11440 \\ 1 & 9 & 45 & 165 & 495 & 1287 & 3003 & 6435 & 12870 & 24310 \\ 1 & 10 & 55 & 220 & 715 & 2002 & 5005 & 11440 & 24310 & 48620 \end{pmatrix}$$

$$X = \begin{pmatrix} 10 & -45 & 120 & -210 & 252 & -210 & 120 & -45 & 10 & -1 \\ -45 & 285 & -870 & 1638 & -2058 & 1770 & -1035 & 395 & -89 & 9 \\ 120 & -870 & 2892 & -5754 & 7512 & -6645 & 3970 & -1541 & 352 & -36 \\ -210 & 1638 & -5754 & 11934 & -16083 & 14585 & -8889 & 3507 & -812 & 84 \\ 252 & -2058 & 7512 & -16083 & 22252 & -20626 & 12804 & -5131 & 1204 & -126 \\ -210 & 1770 & -6645 & 14585 & -20626 & 19490 & -12305 & 5005 & -1190 & 126 \\ 120 & -1035 & 3970 & -8889 & 12804 & -12305 & 7890 & -3255 & 784 & -84 \\ -45 & 395 & -1541 & 3507 & -5131 & 5005 & -3255 & 1361 & -332 & 36 \\ 10 & -89 & 352 & -812 & 1204 & -1190 & 784 & -332 & 82 & -9 \\ -1 & 9 & -36 & 84 & -126 & 126 & -84 & 36 & -9 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 0.0010 & -0.0045 & 0.0120 & -0.0210 & 0.0252 & -0.0210 & 0.0120 & -0.0045 & 0.0010 & -0.0001 \\ -0.0045 & 0.0285 & -0.0870 & 0.1638 & -0.2058 & 0.1770 & -0.1035 & 0.0395 & -0.0089 & 0.0009 \\ 0.0120 & -0.0870 & 0.2892 & -0.5754 & 0.7512 & -0.6645 & 0.3970 & -0.1541 & 0.0352 & -0.0036 \\ -0.0210 & 0.1638 & -0.5754 & 1.1934 & -1.6083 & 1.4585 & -0.8889 & 0.3507 & -0.0812 & 0.0084 \\ 0.0252 & -0.2058 & 0.7512 & -1.6083 & 2.2252 & -2.0626 & 1.2804 & -0.5131 & 0.1204 & -0.0126 \\ -0.0210 & 0.1770 & -0.6645 & 1.4585 & -2.0626 & 1.9490 & -1.2305 & 0.5005 & -0.1190 & 0.0126 \\ 0.0120 & -0.1035 & 0.3970 & -0.8889 & 1.2804 & -1.2305 & 0.7890 & -0.3255 & 0.0784 & -0.0084 \\ -0.0045 & 0.0395 & -0.1541 & 0.3507 & -0.5131 & 0.5005 & -0.3255 & 0.1361 & -0.0332 & 0.0036 \\ 0.0010 & -0.0089 & 0.0352 & -0.0812 & 0.1204 & -0.1190 & 0.0784 & -0.0332 & 0.0082 & -0.0009 \\ -0.0001 & 0.0009 & -0.0036 & 0.0084 & -0.0126 & 0.0126 & -0.0084 & 0.0036 & -0.0009 & 0.0001 \end{pmatrix}$$

(**Note.** *Differences in values of X and M are caused by the fact that Matlab function inv() performs the matrix inversion using floating-point computations. In fact these matrices have equal elements just multiplied by some scalar, which does not affect their correctness.*)

Results:

Tablica 7: Inverse errors

Right residual error	Left residual error	Total error
0	0	$1.0096 \cdot 10^{-10}$

Tablica 8: LU-decomposition errors

Lower error	Upper error
0	0.9931

Tablica 9: $cond(A)$ & Equation errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$4.1552 \cdot 10^{9}$	0	0	$6.1076 \cdot 10^{-8}$

4. $A = hilb(10)$ and $X = A^{-1}$

$$A = \begin{pmatrix} 1.0000 & 0.5000 & 0.3333 & 0.2500 & 0.2000 & 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 \\ 0.5000 & 0.3333 & 0.2500 & 0.2000 & 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 \\ 0.3333 & 0.2500 & 0.2000 & 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 \\ 0.2500 & 0.2000 & 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 \\ 0.2000 & 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 \\ 0.1667 & 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 & 0.0667 \\ 0.1429 & 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 & 0.0667 & 0.0625 \\ 0.1250 & 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 & 0.0667 & 0.0625 & 0.0588 \\ 0.1111 & 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 & 0.0667 & 0.0625 & 0.0588 & 0.0556 \\ 0.1000 & 0.0909 & 0.0833 & 0.0769 & 0.0714 & 0.0667 & 0.0625 & 0.0588 & 0.0556 & 0.0526 \end{pmatrix}$$

$$X = \begin{pmatrix} 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 \\ -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0002 & 0.0005 & -0.0008 & 0.0008 & -0.0004 & 0.0001 \\ 0.0000 & -0.0000 & 0.0001 & -0.0010 & 0.0043 & -0.0112 & 0.0178 & -0.0166 & 0.0085 & -0.0018 \\ -0.0000 & 0.0000 & -0.0010 & 0.0082 & -0.0379 & 0.1010 & -0.1616 & 0.1529 & -0.0788 & 0.0171 \\ 0.0000 & -0.0002 & 0.0043 & -0.0379 & 0.1767 & 0.4772 & 0.7713 & -0.7358 & 0.3821 & -0.0832 \\ -0.0000 & 0.0005 & -0.0112 & 0.1010 & -0.4772 & 1.3015 & -2.1210 & 2.0377 & -1.0643 & 0.2330 \\ 0.0000 & -0.0008 & 0.0178 & -0.1616 & 0.7713 & -2.1210 & 3.4805 & -3.3638 & 1.7660 & -0.3884 \\ -0.0000 & 0.0008 & -0.0166 & 0.1529 & -0.7358 & 2.0377 & -3.3638 & 3.2677 & -1.7232 & 0.3804 \\ 0.0000 & -0.0004 & 0.0085 & -0.0788 & 0.3821 & -1.0643 & 1.7660 & -1.7232 & 0.9123 & -0.2021 \\ -0.0000 & 0.0001 & -0.0018 & 0.0171 & -0.0832 & 0.2330 & -0.3884 & 0.3804 & -0.2021 & 0.0449 \end{pmatrix}$$

$$M = \begin{pmatrix} 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0000 \\ -0.0000 & 0.0000 & -0.0000 & 0.0000 & -0.0002 & 0.0005 & -0.0008 & 0.0008 & -0.0004 & 0.0001 \\ 0.0000 & -0.0000 & 0.0001 & -0.0010 & 0.0043 & -0.0112 & 0.0178 & -0.0166 & 0.0085 & -0.0018 \\ -0.0000 & 0.0000 & -0.0010 & 0.0082 & -0.0379 & 0.1010 & -0.1616 & 0.1529 & -0.0788 & 0.0171 \\ 0.0000 & -0.0002 & 0.0043 & -0.0379 & 0.1767 & -0.4772 & 0.7712 & -0.7358 & 0.3820 & -0.0832 \\ -0.0000 & 0.0005 & -0.0112 & 0.1010 & -0.4772 & 1.3014 & -2.1207 & 2.0375 & -1.0642 & 0.2330 \\ 0.0000 & -0.0008 & 0.0178 & -0.1616 & 0.7712 & -2.1207 & 3.4802 & -3.3635 & 1.7658 & -0.3883 \\ -0.0000 & 0.0008 & -0.0166 & 0.1529 & -0.7358 & 2.0375 & -3.3635 & 3.2674 & -1.7230 & 0.3804 \\ 0.0000 & -0.0004 & 0.0085 & -0.0788 & 0.3820 & -1.0642 & 1.7658 & -1.7230 & 0.9122 & -0.2021 \\ -0.0000 & 0.0001 & -0.0018 & 0.0171 & -0.0832 & 0.2330 & -0.3883 & 0.3804 & -0.2021 & 0.0449 \end{pmatrix}$$

where all elements of X and M should be multiplied by 10^{12}.

Results:

Tablica 10: Inverse errors

Right residual error	Left residual error	Total error
$9.8757 \cdot 10^{-18}$	$2.2635 \cdot 10^{-16}$	$2.1385 \cdot 10^{-5}$

Tablica 11: LU-decomposition errors

Lower error	Upper error
0.2885	456.1225

Tablica 12: $cond(A)$ & Equation errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$1.6025 \cdot 10^{13}$	$1.5185 \cdot 10^{-15}$	$9.4759 \cdot 10^{-29}$	$1.1095 \cdot 10^{-12}$

4 Conclusions

As we can see, calculating inverse of matrix A, where $A \in \mathbb{R}^{n \times n}$ and is symmetric and positive definite by the Crout method ($A = LU$) is an accurate method and does not produce big errors which is to be expected. We obtained the biggest error when using Hilbert's 10 by 10 matrix, which is predictable as it is a matrix with relatively big conditional number ($cond(hilb(10))$ was in 10^{13} range). Both Forward and Backward stability errors are small or non-existing in every case, moreover as long as the condition number was reasonably small, the relative error was also very small or non-existent. The same applies to LU-decomposition error, Right, Left residual errors and Total error of computing the inverse - they were either zero or of negligible value, provided the condition number was relatively small. All of this together suggests that the Crout method is quite accurate and suitable for such calculations, especially when we assume that our input matrix must be symmetric and positive definite, for which Crout algorithm is perfectly stable.

5 Source Codes

Menu_Crout.m:

```
% MENU
clear
clc
finish=8;
control=1;

%default value
A = [1,-3,2; -3,10,-5; 2,-5,6];

while control~=finish

    control=menu('Menu', 'Input A', 'Display variables',...
    'Compute inverse of A','Find LU-factorization of A',...
    'Compute inverse of U','Compute XL = U^(-1)',...
    'Run Tests','FINISH');

    switch control
        case 1
            clearvars variables
            A=input('A = ');
            while check_matrix(A) == 0
                disp('Matrix is not positive definite');
                A=input('A = ');
            end

        case 2
            disp('A = ');disp(A)

        case 3
            In = crout_inverse(A);
            disp('Inverse of A = ');disp(In)

        case 4
            [L, U] = Crout(A);
            disp('LU-factorization of A = ');disp('L = ');
            disp(L);disp('U = ');disp(U)

        case 5
            [L, U] = Crout(A);
            u1 = invupp(U);
```

```
            disp('Inverse of U = ');disp(u1)

        case 6
            [L,U] = Crout(A);
            X = invupp(U)*invlower(L);
            disp('Solution X = '); disp(X);

        case 7
            CroutErrors;

        case 8
            disp('FINISH')

    end
end
```

Crout.m:

```
function [L, U]=Crout(A)
%function for Crout's method for LU-decomposition
%clc
[n, m]=size(A);
if(n~=m)
    disp('Error! Matrix sizes are not equal')
    return
end
if det(A)==0
    disp('Error! Matrix is singular')
    return
end

L=zeros(size(A));
U=zeros(size(A));

for i = 1:n
    L(i, 1) = A(i, 1);
    U(i, i) = 1;
end

for j = 2:n
    U(1, j) = A(1, j) / L(1, 1);
end

for i = 2:n
    for j = 2:i
```

```
        L(i, j) = A(i, j) - L(i, 1:j - 1) * U(1:j - 1, j);
    end
    for j = i - 1:n
        U(i, j) = (A(i, j) - L(i, 1:i - 1) * U(1:i - 1, j)) / L(i, i);
    end
end

end
```

crout_inverse.m:

```
function [X] = crout_inverse(A)

[m,n] = size(A);
if m~=n
    disp('m should be equal to n');
    return
end

d = diag(A);
if ~all(d)
    disp('Diagonal elements of A equals 0');
    return
end

[L, U] = Crout(A);
B = invlower(L);
C = invupp(U);
X = C*B;

end
```

invlower.m:

```
function X = invlower(A)

[m,n] = size(A);
if m~=n
    disp('m should be equal to n');
    return
end

X = zeros(n);
d = diag(A);
```

```
if ~all(d)
    disp('Diagonal elements of A equals 0');
    return
end


for j=1:n
   X(j,j)=1/A(j,j);
    for i=j+1:n
        s=0;
        for p=i-1:-1:j
            s=s+A(i,p)*X(p,j);
        end
        X(i,j)= -s/A(i,i);
    end
end
end
```

invupp.m:

```
function X = invupp(A)

[m,n] = size(A);
if m~=n
    disp('m should be equal to n');
    return
end

X = zeros(n);
d = diag(A);

if ~all(d)
    disp('Diagonal elements of A equals 0');
    return
end

for j=1:n
   X(j,j)=1/A(j,j);
    for i=j-1:-1:1
        s=0;
        for p=i+1:j
            s=s+A(i,p)*X(p,j);
        end
        X(i,j)= -s/A(i,i);
    end
```

```
end
end
```

check_matrix.m:

```
function b = check_matrix(A)
%function checks if input matrix A is
%symmetric and positive definite

b = issymmetric(A);
if b == 0
    disp('Matrix is not symmetric!');
    return;
end

d = eig(A);
b = all(d > 0);

end
```

CroutErrors.m:

```
function CroutErrors
%Crout algorythm Error check
clear
clc
in = input('Choose matrix(1 - pascal(3), 2 - pascal(5)',...
'3 - pascal(10), 4 - hilb(10))');

if(in==1)
    n=3;
    A=pascal(n);
elseif(in==2)
    n=5;
    A=pascal(n);
elseif(in==3)
    n=10;
    A=pascal(n);
elseif(in==4)
    n=10;
    A=hilb(n);
else
    n=4;
    A=pascal(n);
end
```

```
cond_A=cond(A);
disp('A='); disp(A);
disp('Condition numer of A = '); disp(cond_A);

%Inverse error (1)
I=eye(n);
Y=inv(A); %matlab inverse function
X=crout_inverse(A); %crout inverse
a1=inv(X);
r_R=norm(A*X-I)/(norm(A)*norm(X)); % Right residual error
r_L=norm(X*A-I)/(norm(A)*norm(X)); % Left residual error
error_1=norm(a1-A)/norm(A);

disp('Matlab inv(A)='); disp(Y);
disp('Crout inverse(A)='); disp(X);
disp('Right residual error = '); disp(r_R);
disp('Left residual error = '); disp(r_L);
disp('Inverse error = '); disp(error_1);

%LU-decomposition error (2)
[L,U]=Crout(A);
error_low=norm(A-L*L')/norm(A);
error_upp=norm(A-U*U')/norm(A);
disp('Lower error '); disp(error_low);
disp('Upper error '); disp(error_upp);

%Solving system XL = inv(U) error (3)
Z=crout_inverse(A); % The exact solution of our problem.
u1=Z*L;
H=u1*invlower(L);
e1=norm(H-Z)/norm(Z);
e2=e1/cond(A);
e3=norm(u1-A*H)/(norm(A)*norm(H));
disp('Relative error =');disp(e1)
disp('Forward stability error =');disp(e2)
disp('Backward stability error =');disp(e3)

end
```