14 April 2019

Bartek Żyła
Group 2

LU-decomposition of a matrix using Doolittle's method in solving system of linear equations $Ax = b$.

Project 18

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:

$$A = LU$$

Doolittle'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. We then systematically solve for the entries in L and U from the equations that result from the multiplications necessary for $A = LU$

$$\forall i \in \{1, 2, ..., n-1\}$$

$$U_{i,k} = A_{i,k} - \sum_{j=0}^{i} (L_{i,j} \cdot U_{j,k})$$

for k=i,i+1,...,n-1 produces the k^{th} row of U.

$$L_{i,k} = \frac{A_{i,k} - \sum_{j=0}^{i} (L_{i,j} \cdot U_{j,k})}{U_{k,k}}$$

for i=k,k+1,...,n-1 and $L_{i,i} = 1$ produces the k^{th} row of L.

Thanks to this we can solve given system of equations $Ax = b$ more efficiently by solving 2 simpler systems, namely:

1. $LY = b$ through forward substitution
2. $Ux = Y$ via back substitution

2 Program description

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

Menu
Input matrix A
Input vector b
Display variables
Calculate determinant of A
Compute Ax=b
Calculate Errors
FINISH

- If You press button „Input matrix A” or „Input vactor b”, You will be able to input Your own matrix or vector respectively. If You do not - the default arguments will be used.
- Option „Display variables” will simply display previously input variables.
- After clicking „Calculate determinant of A” the matrix determinant using LU-decomposition will be calculated.
- Option „Compute Ax=b” will solve the system of linear equations $Ax = b$ by Doolittle's algorythm for LU-decomposition.
- „Calculate Errors” will ask for one of 3 pre-defined input for error calculation.
- At last „FINISH” will end the program.

MATLAB functions used:

1. Menu_Doolittle.m - script for graphic interface for the rest of the functions.
2. Doolittle.m - function strictly for computing LU-decomposition from given matrix A using formulas described at the beggining.
3. DoolittleErrors.m - function calculating 3 types of errors for one of 3 selected example.

4. Lower_triangular1.m & Upper_triangular1.m - functions from laboratories used for computing solution for Ax=b where A is lower and upper triangular matrix respectively.

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

3 Numerical tests

All the numerical tests are included in the DoolittleErrors.m file. It works by implementing a simple menu in which the user can choose some specified example. For each test the script sets different A, b and z where A is the usual matrix and b is an vector that we pass to our *Doolittle* function and z is the exact solution. Then the program calls *Doolittle* function and calculates errors as well as the condition number for A.

1. Relative error

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

2. Forward stability error

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

3. Backward stability error

$$\frac{||B - AX||}{||A||||X||}$$

Tests: x is the result we obtain using our implemented *Doolittle* function, whereas z is the predefined solution wich was used to get vector b by multiplication $b = A * z$.

1. $A = pascal(10)$ and $z = 10 * ones(10, 1)$

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

Result:

$$x = \begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \\ 10 \\ 10 \\ 10 \\ 10 \\ 10 \\ 10 \end{pmatrix}$$

Tablica 1: 1st example $cond(A)$ & Errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$4.1552 \cdot 10^9$	0	0	0

2. $A = pascal(15)$ and $z = 15 * ones(15,1)$

$$A = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
1 & 3 & 6 & 10 & 15 & 21 & 28 & 36 & 45 & 55 & 66 & 78 & 91 & 105 & 120 \\
1 & 4 & 10 & 20 & 35 & 56 & 84 & 120 & 165 & 220 & 286 & 364 & 455 & 560 & 680 \\
1 & 5 & 15 & 35 & 70 & 126 & 210 & 330 & 495 & 715 & 1001 & 1365 & 1820 & 2380 & 3060 \\
1 & 6 & 21 & 56 & 126 & 252 & 462 & 792 & 1287 & 2002 & 3003 & 4368 & 6188 & 8568 & 11628 \\
1 & 7 & 28 & 84 & 210 & 462 & 924 & 1716 & 3003 & 5005 & 8008 & 12376 & 18564 & 27132 & 38760 \\
1 & 8 & 36 & 120 & 330 & 792 & 1716 & 3432 & 6435 & 11440 & 19448 & 31824 & 50388 & 77520 & 116280 \\
1 & 9 & 45 & 165 & 495 & 1287 & 3003 & 6435 & 12870 & 24310 & 43758 & 75582 & 125970 & 203490 & 319770 \\
1 & 10 & 55 & 220 & 715 & 2002 & 5005 & 11440 & 24310 & 48620 & 92378 & 167960 & 293930 & 497420 & 817190 \\
1 & 11 & 66 & 286 & 1001 & 3003 & 8008 & 19448 & 43758 & 92378 & 184756 & 352716 & 646646 & 1144066 & 1961256 \\
1 & 12 & 78 & 364 & 1365 & 4368 & 12376 & 31824 & 75582 & 167960 & 352716 & 705432 & 1352078 & 2496144 & 4457400 \\
1 & 13 & 91 & 455 & 1820 & 6188 & 18564 & 50388 & 125970 & 293930 & 646646 & 1352078 & 2704156 & 5200300 & 9657700 \\
1 & 14 & 105 & 560 & 2380 & 8568 & 27132 & 77520 & 203490 & 497420 & 1144066 & 2496144 & 5200300 & 10400600 & 20058300 \\
1 & 15 & 120 & 680 & 3060 & 11628 & 38760 & 116280 & 319770 & 817190 & 1961256 & 4457400 & 9657700 & 20058300 & 40116600
\end{pmatrix}$$

$$z = \begin{pmatrix} 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \end{pmatrix}$$

Result:

$$x = \begin{pmatrix} 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \\ 15 \end{pmatrix}$$

Tablica 2: 2nd example $cond(A)$ & Errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$2.8397 \cdot 10^{15}$	0	0	0

3. $A = hild(10)$ and $z = ones(10, 1)$

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

$$z = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Result:

$$x = \begin{pmatrix} 1.0000 \\ 1.0000 \\ 1.0000 \\ 1.0000 \\ 0.9998 \\ 1.0004 \\ 0.9993 \\ 1.0007 \\ 0.9996 \\ 1.0001 \end{pmatrix}$$

Tablica 3: 3rd example $cond(A)$ & Errors

$cond(A)$	Relative error	Fwd stability error	Back stability error
$1.6025 \cdot 10^{13}$	$3.5784 \cdot 10^{-4}$	$2.2330 \cdot 10^{-17}$	$7.4983 \cdot 10^{-17}$

4 Conclusions

As we can see, solving equations of type $Ax = b$, where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^{n\times 1}$ by the Doolittle method ($A = LU$) is an accurate mathod and does not produce big errors for matrices with $eps \cdot cond(A) < 1$ 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) and it has more complicated 1^{st} column then the other examples. Both Forward and Backward stability errors are small or non-existing in every case, moeover as long as the condition number was reasonably small, the relative error was also very small or non-existant. All of this together suggests that the method is quite accurate.

5 Source Codes

Menu_Doolitttle.m:

```
% MENU
clear
clc
finish=7;
control=1;

%default data %
A=[1,-3,2;-3,10,-5;2,-5,6];
b=[3;-8;8];

while control~=finish

    control=menu('Menu', 'Input matrix A', 'Input vactor b',...
                'Display variables','Calculate determinant of A',...
                'Compute Ax=b', 'Calculate Errors', 'FINISH');

    switch control
        case 1
            A=input('A=');

        case 2
            b=input('b=');

        case 3
            disp('A=');disp(A)
            disp('b=');disp(b)

        case 4
            [L, U] = Doolittle(A);
            u = diag(U);
            det_A = prod(u);
            disp('Determinant of A=');disp(det_A)

        case 5
            [L, U] = Doolittle(A);
            y=Lower_triangular1(L,b);
            x=Upper_triangular1(U,y);
            disp('x=');disp(x)

        case 6
```

```
            DoolittleErrors;

        case 7
            disp('FINISH')
    end
end
```

Doolittle.m:

```
function [L, U]=Doolittle(A)
%function for Doolittle's method for LU-decomposition
clc
[n, m]=size(A);
if(n~=m)
    disp('Error! Matrix sizes are not equal')
    return
end
L=zeros(size(A));
U=zeros(size(A));
for i=1:n
    %upper
    for k=i:n
        sum=0;
        for j=1:i-1
            sum = sum + L(i,j)*U(j,k);
        end
        U(i,k) = A(i,k) - sum;
    end
    %lower
    for k=i:n
        if(i==k)
            L(i,i)=1;
        else
            sum=0;
            for j=1:i-1
                sum = sum + L(k,j)*U(j,i);
            end
            if(U(i,i)==0)
                disp('Error! Division by 0')
                return
            end
            L(k,i) = (A(k,i)-sum)/U(i,i);
        end
    end
end
end
```

DoolittleErrors.m:

```
function DoolittleErrors
%Doolittle algorythm Error check
clear
clc
in = input('Choose example(1 - pascal(10), 2 - pascal(15)',...
    'or 3 - hilb(10))');
if(in==1)
    %Example 1
    A=pascal(10);
    z=10*ones(10,1);
elseif(in==2)
    %Example 2
    A=pascal(15);
    z=15*ones(15,1);
else
    %Example3
    A=hilb(10);
    z=ones(10,1);
end

b=A*z;
[L, U] = Doolittle(A);
y=Lower_triangular1(L,b);
x=Upper_triangular1(U,y);

if(eps*cond(A) > 1)
    disp('eps*cond(A) > 1')
    return
end
e1=norm(x-z)/norm(z);
e2=norm(x-z)/(norm(z)*cond(A));
e3=norm(b-A*x)/(norm(A)*norm(x));
disp('relative error =');disp(e1)
disp('forward stability error =');disp(e2)
disp('backward stability error =');disp(e3)
end
```

Lower_triangular1.m:

```
function [x]=Lower_triangular1(A,b)
% [x]=Lower_triangular1(A,b)
% x is the solution of Ax=b, where A is lower triangular

[m,n]= size(A);
x=zeros(n,1);
```

```
if m~=n,
    disp('m should be equal to n');
    return;
end

if norm(A-tril(A),'fro')>0
    disp('A is not lower triangular!');
    return;
end

d=diag(A);

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

x(1)=b(1)/A(1,1);

for i=2:n,
    s=b(i);
    for j=1:i-1,
        s=s-A(i,j)*x(j);
    end
    x(i)=s/A(i,i);
end
end
```

Upper_triangular.m:

```
function [x]=Upper_triangular1(A,b)
% [x]=Upper_triangular1(A,b)
% x is the solution of Ax=b, where A is upper triangular

[m,n]= size(A);
x=zeros(n,1);

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

if norm(A-triu(A),'fro')>0
    disp('A is not upper triangular!');
    return;
```

```
end

d=diag(A);

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

x(n)=b(n)/A(n,n);

for i=n-1: -1:1,
    s=b(i);
    for j=i+1:n,
        s=s-A(i,j)*x(j);
    end
    x(i)=s/A(i,i);
end
end
```