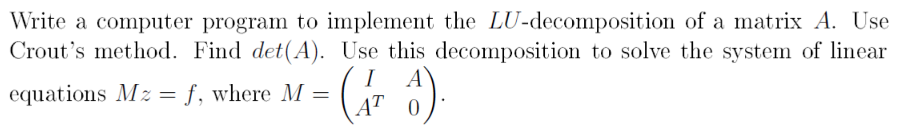
LU-decomposition of matrices using the Crout's method

Szymon Górski

1 Assignment

1.1 Content

The goal of the project was to solve the following task:



Write a computer program to implement the LU-decomposition of a matrix A. Use Crout's method. Find $det(A)$. Use this decomposition to solve the system of linear equations $Mz = f$, where $M = \begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix}$.

I prepared the solutions in MATLAB.

1.2 The Crout's Algorithm

The core of my solution, that is the LU decomposition is done using The Crout's Algorithm. Assuming, that for matrix A of size $n \times n$ the decomposition $A = LU$ exists, the algorithm follows these steps:

- Create U identity matrix and L zero matrix, both of size $n \times n$

```
for k = 1, ..., n
    • for i = k, ..., n
      Compute $l_{ik} = a_{ik} - \sum_{j=1}^{k-1} l_{ij} u_{jk}$
    • for i = k+1, ..., n
      Compute $u_{ki} = (a_{ki} - \sum_{j=1}^{k-1} l_{kj} u_{ji}) / l_{kk}$
```

2 Solutions

2.1 Calculating LU matrices

The implementation of The Crout's Algorithm is presented below. The codes shown in this section don't include all the errors handling present in the project.

```
[L,U] = CroutDecomposition(A)
    A_size = size(A);
    C = min(A_size(1), A_size(2));
    L = zeros(C);
    U = eye(C);
    for i = 1:C
        L(i,1) = A(i,1);
    end
    for j = 2:C
        U(1,j) = A(1,j) / L(1,1);
    end
    for i = 2:C
        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:C
            U(i,j) = (A(i,j) - L(i,1:i-1) * U(1:i-1,j))
                / L(i,i);
        end
    end
```

2.2 Calculating $det(A)$

In this solution all numbers on the diagonal of U are equal to one, therefore it is enough to apply this formula to calculate $det(A)$ for A of size $n \times n$, having done the LU decomposition:

$$det(A) = det(LU) = det(L) * det(U) = det(L) * 1 = \prod_{i=1}^{n} a_{ii} \tag{1}$$

2.3 Calculating $Mz = f$

Assuming that:

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

one can see that:

$$Mz = f \Rightarrow LUz = f \Rightarrow L(Uz) = f \tag{3}$$

It is enough to solve these two linear systems with triangular matrices:

$$Ly = f, Uz = y \tag{4}$$

The implementation:

```
[z] = SolveSystem(M, f)
    f_size = size(f);
    n = f_size(1);
    y = zeros(n,1);
    y(1) = f(1)/L(1,1);
    for i = 2:n
        y(i) = f(i);
        for j = 1:i-1
            y(i) = y(i) - y(j)*L(i,j);
        end
        y(i) =  y(i) / L(i,i);
    end
    z = zeros(n,1);
    z(n) = y(n) / U(n,n);
    for i = n-1:-1:1
        z(i) = y(i);
        for j = n:-1:i+1
            z(i) = z(i) - z(j)*U(i,j);
        end
        z(i) = z(i) / U(i,i);
    end
```

3 Exceptions

3.1 Non-square matrices

The Crout's Algorithm works for non-square matrices, decomposing the biggest submatrix of the given matrix. In further calculations, additional columns and rows could be technically ignored, but this can be confusing, especially when the input matrix has more columns than rows. Therefore, I decided to allow non-square matrices in `CroutDecomposition` function and reject them in `SolveSystem`.

3.2 Matrices without an *LU* decomposition

For a square matrix it is enough to be non-singular (i.e. $det \neq 0$) and have all leading principal minors non-zero to have an *LU* decomposition. (1) It is enough to calculate these values and check if they are not equal to zero. However, because of physical limitations of precision of calculations, MATLAB calculates values with a precision of 16 digits by default. (2) It is common to obtain non-zero values close to zero instead of zeros. Therefore, in calculations of matrix criteria I recognise values with the absolute value smaller than 10^{-12} as a zero.

3.3 Dividing by zero

Zeros cannot be placed on diagonals (otherwise, the function would have thrown an error, as the determinant of such a matrix is zero). Therefore, any occurrence of a zero in this position is caused by inaccuracy of calculations. MATLAB doesn't return an error while dividing by 0, but returns ∞ as a result. This is not a satisfactory solution while calculating linear systems, therefore I added proper exceptions to `CroutDecomposition` and `SolveSystem` functions.

4 Project details

4.1 Interim Report questions

These are answers to questions given in the interim report:

1. How to efficiently recognise matrices without a particular solution?

   $\rightarrow$ See *Matrices without an LU decomposition.*

2. What simplifications can be done while calculating matrices with symmetry axis?

   For more efficient calculation of the M matrix, one can use the Cholesky–Crout algorithm, which allows to perform an $A = LL^T$ decomposition on symmetric positive definite matrices. Indeed, for any matrix A, matrix M (created in a given way) is symmetric. However, the Cholesky–Crout algorithm returns different values in cells on the diagonals of triangular matrices, therefore they cannot be applied for $det(A)$ calculation in this project. And, essentially, is not an algorithm I was supposed to use.

3. What should be the size limit for the input matrix?

   As my project runs in MATLAB, the computation is limited by this software. MATLAB handles issues with too big input by returning the error "The input was too complicated or too big for MATLAB to parse". Therefore, no additional protections must be added to the code.

References

[1] Horn, Roger A.; Johnson, Charles R. (1985), "Matrix Analysis"

[2] `https://www.mathworks.com/help/symbolic/increase-precision-of-numeric-calculations.html`