Laboratorium 2

Szymon Twardosz Dominik Jeżów

12 listopada 2023

1 Wprowadzenie

Celem ćwiczenia było zaimplementowanie trzech algorytmów operacji na macierzach. W szczególności były to:

- rekurencyjne odwracanie macierzy;
- rekurencyjna LU faktoryzacja;
- rekurencyjne obliczanie wyznacznika;

Ćwiczenie wykonaliśmy w języku Julia przy użyciu bibliotek: LinearAlgebra, DataFrames, BenchmarkTools, GFlops, Statistics, Plots oraz środowiska Jupyter Notebook

2 Rekurencyjne odwracanie macierzy

2.1 Wzory i pseudokod

Algorytm przyjmuję na wejściu macierz. Jego zadaniem jest znaleźć macierz odwrotną do zadanej. Robi to metodą rekurencyjną, zgodnie z wzorami:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$
$$A^{-1} = \begin{bmatrix} A_{11}^{-1}\left(I + A_{12}S_{22}^{-1}A_{21}A_{11}^{-1}\right) & -A_{11}^{-1}A_{12}S_{22}^{-1} \\ -S_{22}^{-1}A_{21}A_{11}^{-1} & S_{22}^{-1} \end{bmatrix}$$

Pseudokod:

```
function(m: matrix) -> matrix:
 if dim(matrix) == 1x1:
 return 1 / matrix[1, 1]
 else:
 use_formulas_above();
```

2.2 Złożoność obliczeniowa

Nasza implementacja odwracania macierzy, do mnożenia macierzy używa algorytmu Strassena, którego złożoność wynosi: $O(n^{\log_2(7)})$. Złożoność obliczeniowa algorytmu to rozwiązanie równania:

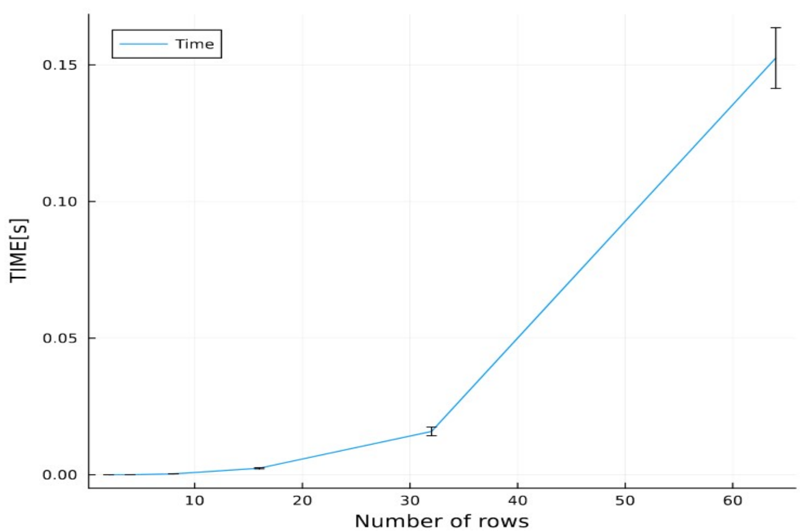
$$O(n) = 2 \cdot O\left(\frac{n}{2}\right) + 10 \cdot O(n^{\log_2(7)}) + d$$

,gdzie n to liczba elementów w jednym wierszu macierzy.
Na podstawie Master Theorem złożoność algorytmu wynosi: $O(n^{\log_2(7)})$

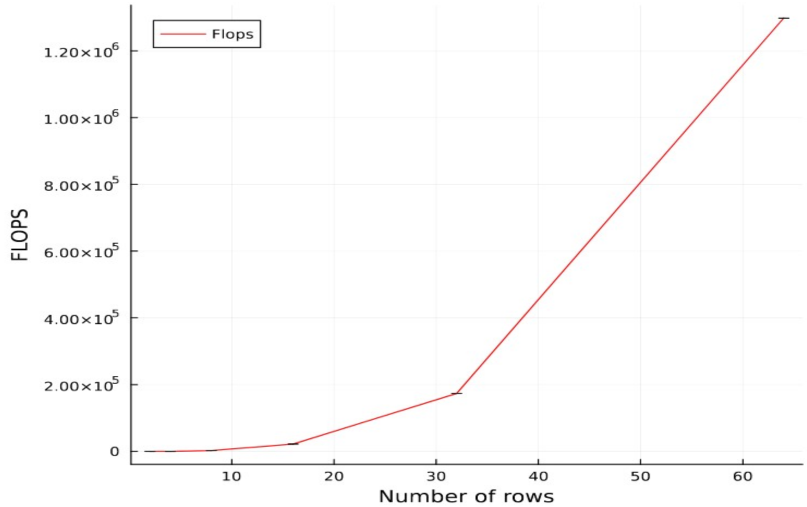
2.3 Kod programu i wykresy

```
function matrix_inverse(A)
    """
    :return matrix B, such that A * B = I
    """

    n = size(A, 1)
    m = size(A, 2)

    if n != m
        throw(NonSquareMatrixError("Matrix is not the square"))
    end

    if n == 1
        return [1 / A[1, 1]]
    end


    m = div(n, 2)
    A11, A12, A21, A22 = divide_matrix(A, n)

    A11_inversed = matrix_inverse(A11)
    S22 = A22 - strassen(strassen(A21, A11_inversed), A12)
    S22_inversed = matrix_inverse(S22)



    B = zeros(n, n)
    eye = Matrix{Float64}(I, div(n, 2) , div(n, 2))

    B[1:m, 1:m] = strassen(A11_inversed, eye + strassen(strassen(strassen(A12, S22_inversed), A21), A11_inversed))
    B[1:m, m+1:n] = strassen(strassen(-A11_inversed, A12), S22_inversed)
    B[m+1:n, 1:m] = strassen(strassen(-S22_inversed, A21), A11_inversed)
    B[m+1:n, m+1:n] = S22_inversed

    return B

end
```

Wykres 1: Kod algorytmu odwracania macierzy



Wykres 2: Wykres zależności czasu od liczby wierszy odwracanej macierzy

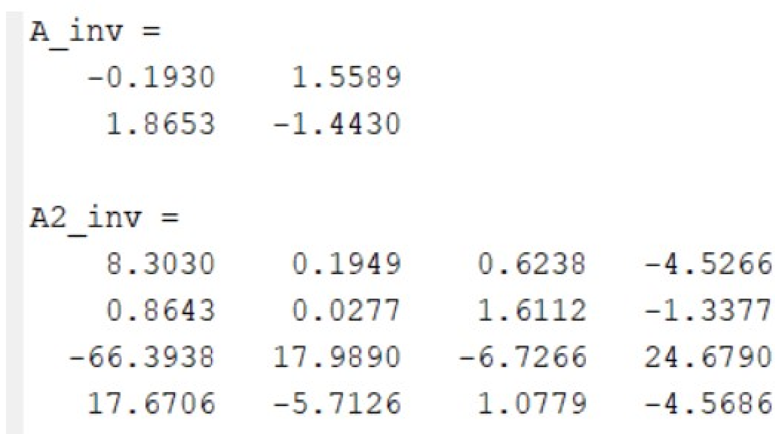


Wykres 3: Wykres zależności operacji zmiennoprzecnikowych od liczby wierszy odwracanej macierzy

2.4 Porównanie wyników z Matlabem

Kod testowy w języku Matlab

```
A = [0.5488, 0.5929;
0.7094, 0.0734];
A_inv = inv(A);
fprintf("A");
disp(A_inv);
fprintf("A2");
A2 = [0.2722, 0.1980, 0.1500, 0.4826;
0.6070, 0.3818, 0.2489, 0.6313;
0.1088, 0.9294, 0.1723, 0.5508;
0.3195, 0.5077, 0.3096, 0.9883];
disp(inv(A2);
```



```
A_inv =
   -0.1930    1.5589
    1.8653   -1.4430

A2_inv =
    8.3030    0.1949    0.6238   -4.5266
    0.8643    0.0277    1.6112   -1.3377
  -66.3938   17.9890   -6.7266   24.6790
   17.6706   -5.7126    1.0779   -4.5686
```

Wykres 4: Wyniki w Matlabie

```
2x2 Matrix{Float64}:
 -0.192995   1.55894
  1.86526   -1.44299

4x4 Matrix{Float64}:
   8.30305    0.194897   0.623801  -4.52664
   0.864319   0.027741   1.61119   -1.33773
 -66.3938    17.989     -6.72664   24.679
  17.6706    -5.71258    1.07787   -4.56863
```

Wykres 5: Wyniki naszego algorytmu

3 Rekurencyjna LU faktoryzacja

3.1 Wzory i pseudokod

Algorytm przyjmuję na wejściu macierz. Jego zadaniem jest znaleźć macierze L i U, takie że L jest trójkątna dolna, U jest trójkątna górna oraz L* U = A. Robi to metodą rekurencyjną, zgodnie z wzorami:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$
$$L = \begin{bmatrix} L_{11} & 0 \\ A_{21} * U_{11}^{-1} & S_L \end{bmatrix}$$
$$U = \begin{bmatrix} U_{11} & L_{11}^{-1} * A_{12} \\ 0 & S_U \end{bmatrix}$$

gdzie:

- S - dopełnienie Schura równe $(A_{22} - U_{11}^{-1} * S_L * L_{11}^{-1} * A_{12})$
- S_L, S_U - są to macierze obliczone przez rekurencyjne odwołanie do faktoryzacji macierzy S
- L_{11}, U_{11} - powstały przez LU faktoryzacje A_{11}

Pseudokod:
```
if size(A) == 2x2
 L = [A[1,1], 0 ; A[2,1], A[2,2] - A[2,1] * A[1,2] / A[1,1]]1
 U = [1, A[1,2]/A[1,1]; 0, 1]
 else:
 działaj według powyższych wzorów
```

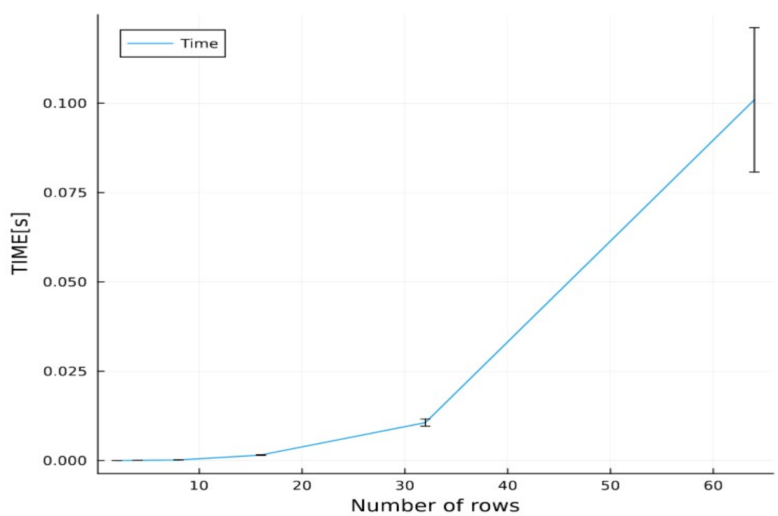
3.2 Złożoność obliczeniowa

Podobnie jak w przypadku odwracania macierzy mamy do czynienia z złożonością obliczeniową $O(n^{\log_2(7)})$

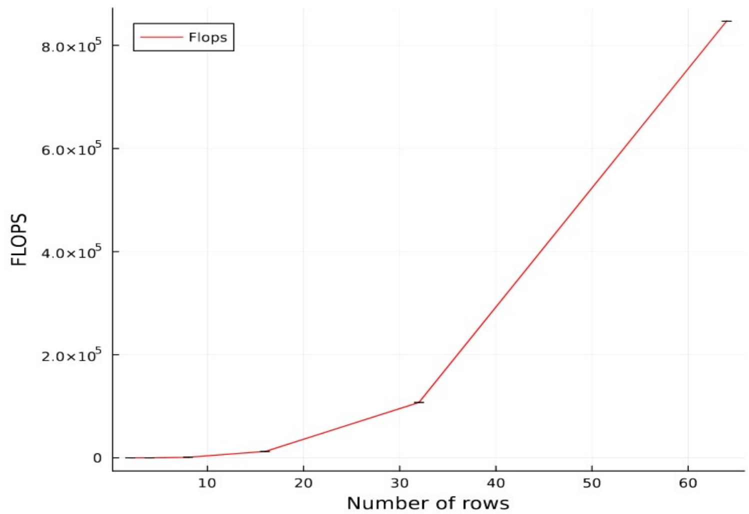
3.3 Kod programu i wykresy

```
function _matrix_LU_factor(A, n)
    if n == 2
        a22 = A[2,2] - A[2,1] * A[1,2] / A[1,1]
        return LowerTriangular([A[1,1] 0; A[2,1] a22]),  UnitUpperTriangular([1 A[1,2]/A[1,1]; 0 1])
    end
    """
    Helping function to matrix_LU_factor() func
    """
    m = div(n, 2)

    A11, A12, A21, A22 = divide_matrix(A, n)
    L11, U11 = _matrix_LU_factor(A11, m)

    U12 = strassen(matrix_inverse(L11),A12)
    L21 = strassen(A21,matrix_inverse(U11))

    S = A22 - strassen(L21, U12)

    L22, U22 = _matrix_LU_factor(S, m)

    L = vcat(hcat(L11, zeros(m,m)), hcat(L21, L22))
    U = vcat(hcat(U11, U12), hcat(zeros(m,m), U22))
    return LowerTriangular(L), UnitUpperTriangular(U)
end


function matrix_LU_factor(A)
    """
    :return matrixes L, U : L * U = A and L is upper triangular matrix and U is lower triangular matrix
    A must be in shape (2^i, 2^i) where i is Natural
    """

    n = size(A, 1)
    m = size(A, 2)

    if n != m
        throw(NonSquareMatrixError("Matrix is not the square"))
    end
    log2_n = log2(n)
    if !(trunc(Int, log2_n) - log2_n ≈ 0)
        throw(ArgumentError("A, not in (2^i, 2^i) shape, shape= $(size(A))"))
    end

    return _matrix_LU_factor(A, n)
end
```

Wykres 6: Kod algorytmu faktoryzacji LU



Wykres 7: Wykres zależności czasu od liczby wierszy odwracanej macierzy

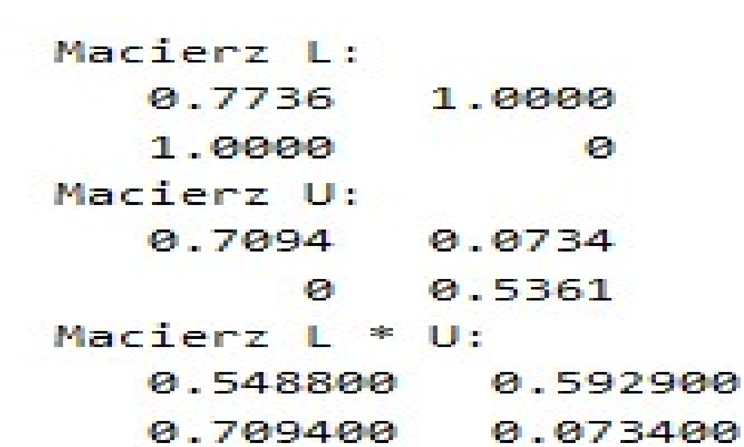


Wykres 8: Wykres zależności operacji zmiennoprzecnikowych od liczby wierszy odwracanej macierzy

3.4 Porównanie wyników z Matlabem

Kod testowy w języku Matlab

```
A = [0.5488, 0.5929;
 0.7094, 0.0734];
[L,U] = lu(A)
fprintf('Macierz L:');
disp(L);
fprintf('Macierz U:');
disp(U);
fprintf('Macierz L * U:');
disp(L * U);
```



Wykres 9: Wyniki w Matlabie

```
2×2 LowerTriangular{Float64, Matrix{Float64}}:
 0.5488    ⋅
 0.7094   -0.693005

2×2 UnitUpperTriangular{Float64, Matrix{Float64}}:
 1.0  1.08036
  ⋅   1.0

2×2 Matrix{Float64}:
 0.5488  0.5929
 0.7094  0.0734

LU{Float64, Matrix{Float64}, Vector{Int64}}
L factor:
2×2 Matrix{Float64}:
 1.0       0.0
 0.773612  1.0
U factor:
2×2 Matrix{Float64}:
 0.7094  0.0734
 0.0     0.536117
```

Wykres 10: Wyniki naszego algorytmu

Otrzymane przez nas macierze nie są takie same poniewrz w Matlabie faktoryzacja LU zwraca L jako **przepermutowaną** macierz trójkątną dolną z jedynkami na przekątnej. Pomimo tego iloczyn macierzy L, U, wyliczonych naszą implementacją faktoryzacji oraz ta wyliczona przez wbudowaną funkcje w Matlab'ie, jest równy macierzy A.

4 Rekurencyjne obliczanie wyznacznika

4.1 Wzory i pseudokod

Algorytm przyjmuję na wejściu macierz. Jego zadaniem jest obliczenie wyznacznika A. Robi to korzystając z rekurencyjnej metody znajdywania LU faktoryzacji:

$$\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix}$$
$$\begin{bmatrix} 1 & u_{12} & \dots & u_{1n} \\ 0 & 1 & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

$det(A) = \prod_{i=1}^{n} l_{ii}$

Pseudokod:

```
L, U = lu_factorisation(A)
diagonal = [L[i,i] for i in 1: L.size]
return prod(diagonal)
```

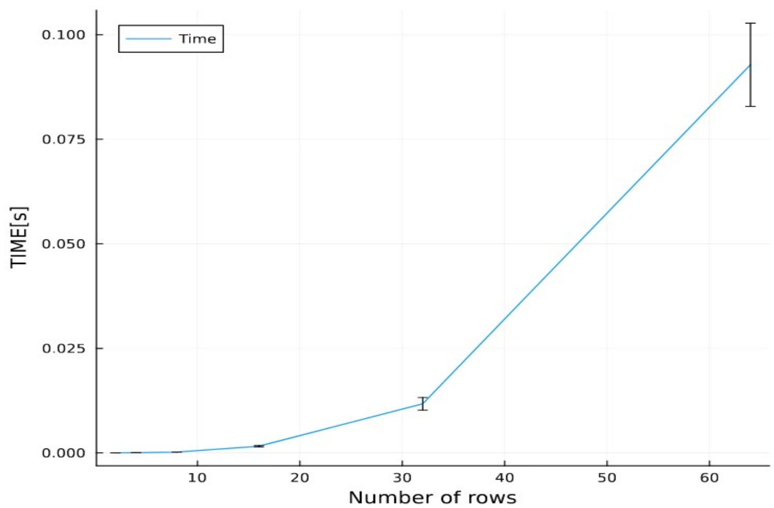
4.2 Złożoność obliczeniowa

Podobnie jak w przypadku odwracania macierzy mamy do czynienia z złożonością obliczeniową $O(n^{\log_2(7)})$

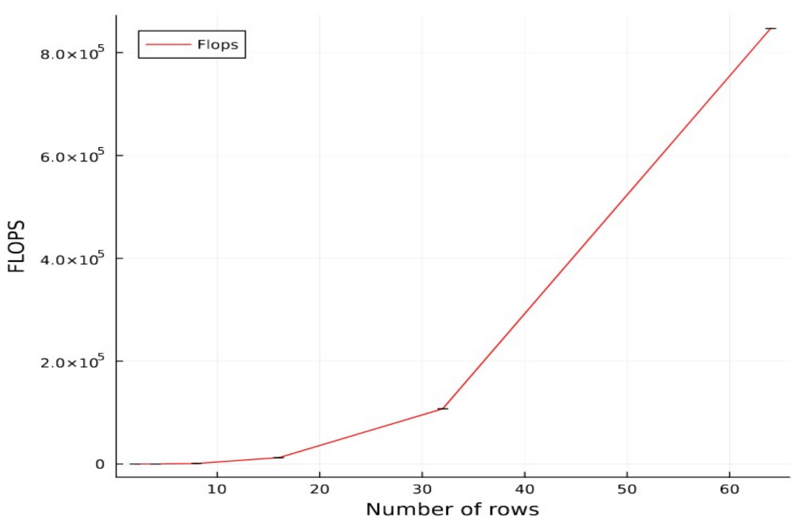
4.3 Kod programu i wykresy

```
function matrix_determinant(A)
    """
    :return determinant of matrix
    A must be in shape (2^i, 2^i) where i is Natural
    """

    L, U = matrix_LU_factor(A)

    diagonal = [L[i,i] for i in 1:size(A, 1)]
    return prod(diagonal)
end
```

Wykres 11: Kod algorytmu faktoryzacji LU



Wykres 12: Wykres zależności czasu od liczby wierszy odwracanej macierzy

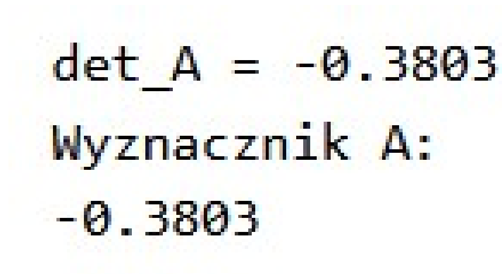


Wykres 13: Wykres zależności operacji zmiennoprzecnikowych od liczby wierszy odwracanej macierzy

4.4 Porównanie wyników z Matlabem

Kod testowy w języku Matlab

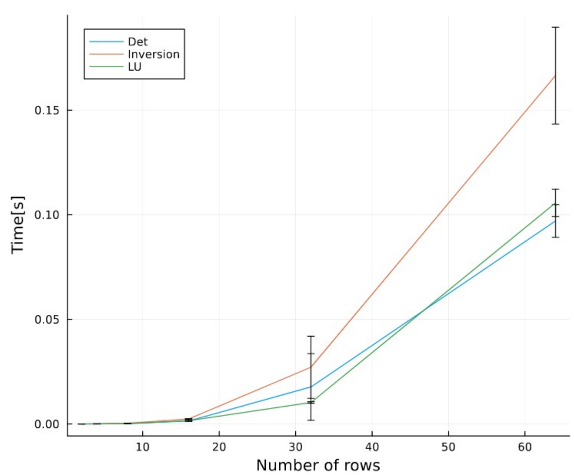
`A = [0.5488, 0.5929;  0.7094, 0.0734];`
`det`$_A = det(A)$
`fprintf('Wyznacznik A:');`
`disp(det`$_A$);



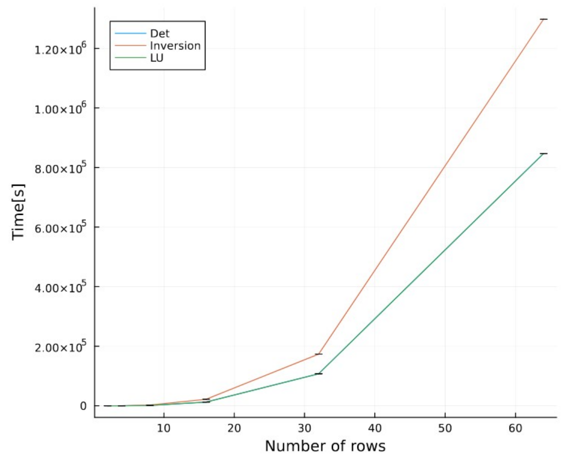
Wykres 14: Wyniki w Matlabie

Wyniki algorytmu zgadzają się z tymi zwróconymi przez Matlaba

5 Porównanie algorytmów



Wykres 15: Porównanie czasu wykonywania



Wykres 16: Porównanie operacji zmiennoprzecinkowych

6 Wnioski

1. Zaimplementowane algorytmy działają poprawnie. Potwierdzają to testy przeprowadzonę przy pomocy programu komputerowego Matlab
2. Dzięki algorytmowi Strassena (mnożenia macierzy) wszystkie nasze implementacje posiadaja złożoność $O(n^{\log_2(7)})$.
3. Algorytmy faktoryzacji LU oraz liczenia wyznacznika korzystają z algorytmu odwracania macierzy. Mimo to potrzebują mniej czasu oraz operacji na liczbach zmiennoprzecinkowych niż rekurencyjna inwersja macierzy. Może być to spowodowane tym, że w metoda faktoryzacji LU (pośrednio też obliczania wyznacznika) wywołują funkcję odwracania macierzy na mniejszych macierzach (częściach zadanych macierzy). Oprócz tego, do faktoryzacji LU wykorzystaliśmy również klasy: LowerTriangular oraz UnitUpperTriangular, które optymalizują działania na macierzach trójkątnych.