AKADEMIA GÓRNICZO-HUTNICZA
Wydział Informatyki

Projekt nr 2
Z przedmiotu Rachunek Macierzowy

"Rozkłady macierzy"

Autor: *Jacek Tyszkiewicz i Michał Godek*
Kierunek studiów: *Informatyka*

Kraków, 2025

Spis treści

1. Opracowanie algorytmów mnożenia macierzy

1.1. Cel projektu

Celem projektu było zaimplementowanie dwóch wersji algorytmów, których są wykorzystywane do rozwiązywania układ równań. Mowa tu o algorytmie eliminacji Gaussa oraz algorytmie LU. W celach badawczych zostały one zaimplementowane w następujących konfiguracjach:

1. Algorytm eliminacji Gaussa bez pivotingu generujący jedynki na przekątnej
2. Algorytm eliminacji Gaussa z pivotingiem
3. Algorytm LU faktoryzacji bez pivotingu
4. Algorytm LU faktoryzacji z pivotingiem

1.2. Algorytm eliminacji Gaussa bez pivotingu generujący jedynki na przekątnej

Celem algorytmu eliminacji Gaussa jest rozwiązanie układu równań liniowych:

$$A \cdot x = b \quad \text{gdzie} \quad A \in \mathbb{R}^{n \times n}, \quad x, b \in \mathbb{R}^n$$

Algorytm składa się z dwóch głównych etapów:

1. Eliminacja w przód (górna trójkątna macierz)

Dla kolejnych wierszy macierzy A dokonujemy operacji:

– Dzielimy wiersz i przez element a_{ii}, aby uzyskać jedynkę na przekątnej:

$$R_i \leftarrow \frac{R_i}{a_{ii}}$$

– Dla każdego wiersza $j > i$ odejmujemy wiersz i przemnożony przez a_{ji}:

$$R_j \leftarrow R_j - a_{ji} \cdot R_i$$

– Te same operacje wykonujemy również na wektorze prawej strony b.

Po zakończeniu tego etapu otrzymujemy układ równań w postaci górnej trójkątnej:

$$U \cdot x = c$$

1.2.1. Podstawianie wsteczne – proste wyjaśnienie

Po przekształceniu macierzy do postaci trójkątnej górnej (czyli z zerami pod przekątną), możemy obliczyć rozwiązania zaczynając od końca.

– Najpierw obliczamy ostatnią niewiadomą x_n, bo jest tylko w jednym równaniu:

$$x_n = \frac{c_n}{a_{nn}}$$

– Potem obliczamy x_{n-1}, podstawiając wcześniej obliczone x_n:

$$x_i = \frac{1}{a_{ii}} \left(c_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) \tag{1.1}$$

```python
def gauss_elimination(A, b):

    A = A.copy().astype(float)
    b = b.copy().astype(float)
    n = len(b)

    # Eliminacja w przód
    for i in range(n):
        # Tworzymy 1 na przekątnej przez dzielenie całego wiersza
        diagonal = A[i, i]
        A[i] = A[i] / diagonal
        b[i] = b[i] / diagonal

        # Eliminacja elementów
        for j in range(i+1, n):
            multiplier = A[j, i]
            A[j] = A[j] - multiplier * A[i]
            b[j] = b[j] - multiplier * b[i]

    # Podstawianie wsteczne
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])).item()

    return x
```

1.3. Algorytm eliminacji Gaussa z pivotingiem

1. Eliminacja w przód z pivotingiem (górna trójkątna macierz)

Dla kolejnych wierszy macierzy A wykonujemy operacje:

- Znajdujemy wiersz z największym modułem elementu w kolumnie i:

$$\text{max\_row} = \arg\max_{i \le k \le n} |a_{ki}|$$

- Jeśli max_row $\neq i$, zamieniamy i-ty i max_row-ty wiersz miejscami zarówno w macierzy A, jak i w wektorze b (pivoting):

$$A_{i,\cdot} \leftrightarrow A_{\text{max\_row},\cdot}, \quad b_i \leftrightarrow b_{\text{max\_row}}$$

- Dzielimy wiersz i przez element a_{ii}, aby uzyskać jedynkę na przekątnej:

$$R_i \leftarrow \frac{R_i}{a_{ii}}$$

- Dla każdego wiersza $j > i$ odejmujemy wiersz i przemnożony przez a_{ji}:

$$R_j \leftarrow R_j - a_{ji} \cdot R_i$$

- Te same operacje wykonujemy również na wektorze prawej strony b.

Po zakończeniu tego etapu otrzymujemy układ równań w postaci górnej trójkątnej:

$$U \cdot x = c$$

1.2.1. Podstawianie wsteczne – z pivotingiem

Po przekształceniu macierzy do postaci trójkątnej górnej możemy wyznaczyć rozwiązanie zaczynając od końca:

- Najpierw obliczamy ostatnią niewiadomą x_n, bo występuje tylko w jednym równaniu:

$$x_n = \frac{c_n}{a_{nn}}$$

- Potem obliczamy x_{n-1}, podstawiając wcześniej obliczone x_n:

$$x_i = \frac{1}{a_{ii}} \left(c_i - \sum_{j=i+1}^{n} a_{ij} \cdot x_j \right) \tag{1.2}$$

```
def gauss_elimination_with_partial_pivoting(A, b):
    A = A.copy().astype(float)
    b = b.copy().astype(float).reshape(-1)
    n = len(b)

    # Eliminacja w przód
    for i in range(n):
        # 1) Znajdź wiersz z największym (modułowo) elementem w
            kolumnie i (od wiersza i do n-1)
        max_row = i + np.argmax(np.abs(A[i:, i]))

        # 2) Zamiana wierszy i <-> max_row (jeśli różne)
        if max_row != i:
            A[[i, max_row], :] = A[[max_row, i], :]
            b[i], b[max_row] = b[max_row], b[i]

        # 3) Wyzerowanie elementów w kolumnie i, w wierszach poniżej
            i
        pivot = A[i, i]
        for j in range(i+1, n):
            factor = A[j, i] / pivot
            A[j, i:] -= factor * A[i, i:]
            b[j] -= factor * float(b[i])

    # Podstawianie wsteczne
    x = np.zeros(n, dtype=float)
    for i in range(n-1, -1, -1):
        x[i] = ((b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]).item
            ()

    return x
```

1.4. Algorytm faktoryzacji LU (bez pivotingu)

Do macierzy która będzie faktoryzowana przystawiamy z lewej strony macierz jednostkową o tym samym wymiarze. Następnie, dla każdej, oprócz ostatniej kolumny macierzy A trzeba wyzerować wartości znajdujące się pod przekątną. Dla każdej wartości

$$a_{w,k}$$

pod przekątną trzeba obliczyć

$$a_{w,k}/a_{k,k}$$

następnie odjąć od danego wiersza powyższy iloraz przemnożony przez wiersz o indeksie bierzącej kolumny. Dla macierzy po lewej stronie wynik dzielenia jest wstawiany w miejsce zgodne z indeksami bez dodatkowych operacji. np. dla macierzy 3x3

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

dostawiamy

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

Dla pierwszej kolumny bierzemy pierwszy element pod przekątną czyli z drugiego rzędu i wykonujemy operacje na wierszach

$$w2 - w1 * a_{2,1}/a_{1,1}$$

co daje rezultat

$$\begin{bmatrix} 1 & 0 & 0 \\ a_{2,1}/a_{1,1} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,2} - a_{1,2} * (a_{2,1}/a_{1,1}) & a_{2,3} - a_{1,3} * (a_{2,1}/a_{1,1}) \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}$$

Następnie obliczamy iloraz dla kolejnego elementu w pierwszej kolumnie i wykonujemy operacje na wierszach

$$w3 - w1 * a_{3,1}/a_{1,1}$$

co daje rezultat

$$\begin{bmatrix} 1 & 0 & 0 \\ a_{2,1}/a_{1,1} & 1 & 0 \\ a_{3,1}/a_{1,1} & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,2} - a_{1,2} * (a_{2,1}/a_{1,1}) & a_{2,3} - a_{1,3} * (a_{2,1}/a_{1,1}) \\ 0 & a_{2,2} - a_{1,2} * (a_{3,1}/a_{1,1}) & a_{2,3} - a_{1,3} * (a_{3,1}/a_{1,1}) \end{bmatrix}$$

Postępujemy dla kolejnych kolumn podobnie aż otrzymamy macierz dolną trójkątną i górną trójkątną o postaci

$$L = \begin{bmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{bmatrix} \qquad U = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a'_{2,2} & a'_{1,3} \\ 0 & 0 & a'_{1,3} \end{bmatrix}$$

wynik mnożenia L*U wynosi macierz A zdefiniowana na początku.

```python
from IPython.display import display

O = np.identity(R)

A_work = A.copy().astype(float)
B_work = B.copy().astype(float)
x1 = np.linalg.solve(A, B)
n = len(A)

# Eliminacja Gaussa (bez pivotingu)
for i in range(n-1):
    for j in range(i+1,n):
        div = (A_work[j,i]/A_work[i,i])
        O[j,i] = div
        A_work[j] -= A_work[i]*div

# Zaokrąglanie zer bliskich 0
A_work[abs(A_work) < 0.00001] = 0.0

# L i U gotowe
display("Macierz L (dolna trójkątna):", O)
display("Macierz U (górna trójkątna):", A_work)
#display("Wektor B:", B)
display("Rozwiązanie z numpy.linalg.solve (kontrola):", x1)

L = O
U = A_work

# mamy L i U - koniec, finito
C = np.zeros((R,1))
C[0] = B_work[0]
for i in range(1,n):
    known_c_sum = 0
    for j in range(0,i):
        known_c_sum += L[i,j] * C[j]
    C[i] = B_work[i]-known_c_sum
```

```python
# Podstawianie wsteczne
x = np.zeros((R,1))
x[n-1,0] = C[n-1,0]/U[n-1,n-1]

for i in range(n-2,-1,-1):
    known_xu_sum = 0
    for j in range(i+1,n):
        known_xu_sum += U[i,j] * x[j,0]
    x[i,0] = (C[i,0]-known_xu_sum)/U[i,i]

# Wyniki końcowe
display("Rozwiązanie x:", x)
display("Sprawdzenie: L @ U @ x =", L @ U @ x)
display("Oryginalny wektor B:", B)
```

1.5. Algorytm LU faktoryzacji z pivotingiem

Niech

$$A = \begin{bmatrix} 0 & 4 & 1 \\ 1 & 1 & 3 \\ 2 & -2 & 1 \end{bmatrix}$$

Dla każdej kolumny k wybieramy wiersz z największym elementem w kolumnie i zamieniamy miejscami, tak samo zamieniamy wiersze w macierzy jednostkowej 'P1' którą zapisujemy na boku.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 4 & 1 \\ 1 & 1 & 3 \\ 2 & -2 & 1 \end{bmatrix}$$

Maksymalna wartość w kolumnie 1 to 2, w trzecim rzędzie. Po zamianie rzędów 1 z 3 otrzymujemy:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 \\ 1 & 1 & 3 \\ 0 & 4 & 1 \end{bmatrix}$$

Po zamianie wierszy przystępujemy do zerowania komórek pod przekątną w danej kolumnie. Do tego potrzebna będzie jeszcze jedna macierz jednostkowa 'P2' dostawiona z boku.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 \\ 1 & 1 & 3 \\ 0 & 4 & 1 \end{bmatrix}$$

Podobnie jak w wersji bez pivotingu dla danej kolumny 'k' dla danego wiersza 'w' wykonujemy działania na wierszach

$$w_w - w_k * a_{w,k}/a_{k,k}$$

zarówno w głównej macierzy jak i macierzy 'P2'. Te operacje robimy tylko dla bieżącej kolumny.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 \\ 0 & 2 & 5/2 \\ 0 & 4 & 1 \end{bmatrix}$$

Przy przejściu do następnej kolumny znowu trzeba dostawić macierz jednostkową, wybrać wiersz z największym elementem, zamienić wiersze, dostawić kolejną macierz jednostkową, wyzerować kolumne pod przekątną. Na koniec mamy ciąg macierzy na przemian jednostkowych z zamienionymi wierszami oraz macierz jednostkowych, każda z pojedyńczą kolumną pod przekątną wypełnioną wartościami.

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 \\ 0 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Ostatnim krokiem jest pomnożenie macierzy w następujący sposób - bierzemy przedostatnią macierz i mnożymy co drugą macierz w ciągu macierz. Wynik oznaczamy B

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1/2 & 1/2 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

- mnożymy teraz macierze jednostkowe w których tylko zmienialiśmy rzędy. Wynik oznaczamy P

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Macierz dolna trójkątna wynosi P*B

$$L = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1/2 & 1/2 & 1 \end{bmatrix}$$

Macierz górna trójkątna to ostatnia macierz w powstałym ciągu.

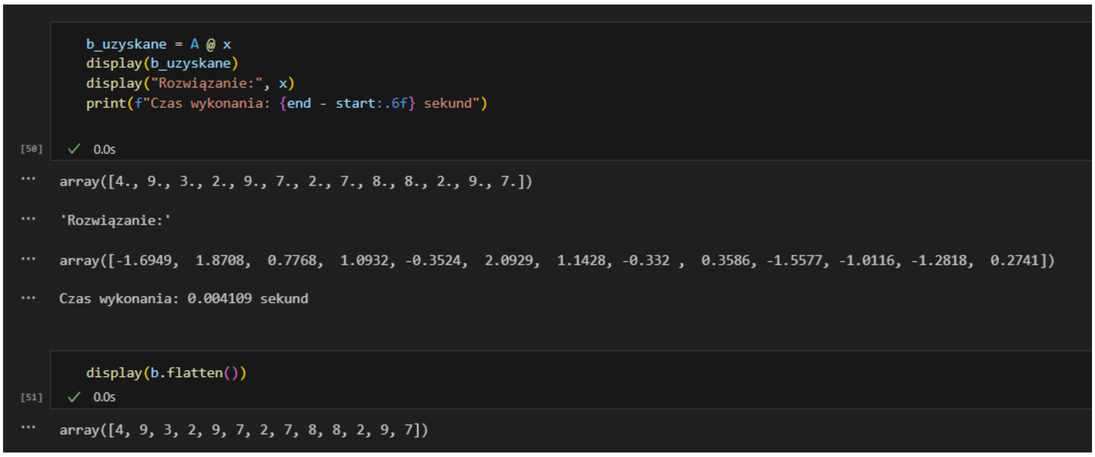
$$U = \begin{bmatrix} 2 & -2 & 1 \\ 0 & 4 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

```python
import numpy as np
from IPython.display import display

def lu_no_pivot(A):
    n = A.shape[0]
    L = np.identity(n)
    U = A.copy().astype(float)

    for i in range(n-1):
        for j in range(i+1,n):
            div = (U[j,i]/U[i,i])
            L[j,i] = div
            U[j] -= U[i]*div
    return L, U


def lu_pivot(A):
    n = A.shape[0]
    A = A.copy().astype(float)

    arr = [np.identity(n)]
    U = A.copy()

    for i in range(n-1):
        P1 = np.identity(n)
        m = np.argmax(np.abs(U[i:,i]), axis=0) + i  # pivot wybierany
            na podstawie |elementów|
        U[[i, m]] = U[[m, i]]  # zamiana wierszy
        P1[[i, m]] = P1[[m, i]]
        arr.append(P1)

        P2 = np.identity(n)
        for j in range(i+1,n):
            div = (U[j,i]/U[i,i])
            P2[j,i] = div
            U[j] -= U[i]*div
        arr.append(P2)

    P = np.identity(n)
```

```
    for i in range(len(arr)-2, -1, -2):
        P = P @ arr[i]

    L = np.identity(n)
    for i in range(1, len(arr), 1):
        L = L @ arr[i]
    L = P @ L

    return P, L, U


"""A_screen = np.array([
    [0, 4, 1],
    [1, 1, 3],
    [2, -2, 1]
], dtype=float)
"""
P, L, U = lu_pivot(A)

display("Macierz A (oryginalna):", A)
display("Macierz permutacji P:", P)
display("Macierz dolna L:", L)
display("Macierz górna U:", U)

display("Weryfikacja: P @ A =")
display(P @ A)
display("Weryfikacja: L @ U =")
display(L @ U)
```

1.6. Wyniki działania algorytmów

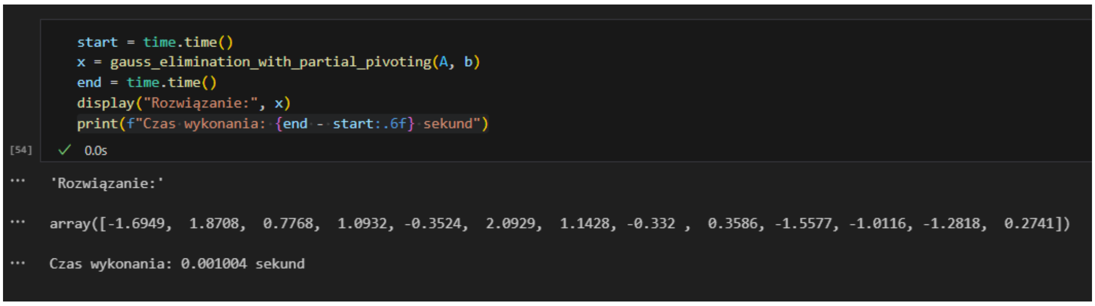
1.7. Algorytm eliminacji Gaussa bez pivotingu generujący jedynki na przekątnej



```
b_uzyskane = A @ x
display(b_uzyskane)
display("Rozwiązanie:", x)
print(f"Czas wykonania: {end - start:.6f} sekund")
```
```
[50] ✓ 0.0s
... array([4., 9., 3., 2., 9., 7., 2., 7., 8., 8., 2., 9., 7.])
... 'Rozwiązanie:'
... array([-1.6949,  1.8708,  0.7768,  1.0932, -0.3524,  2.0929,  1.1428, -0.332 ,  0.3586, -1.5577, -1.0116, -1.2818,  0.2741])
... Czas wykonania: 0.004109 sekund
```
```
display(b.flatten())
```
```
[51] ✓ 0.0s
... array([4, 9, 3, 2, 9, 7, 2, 7, 8, 8, 2, 9, 7])
```

Rys. 1.1. wyniki

1.8. Algorytm eliminacji Gaussa z pivotingiem



```
start = time.time()
x = gauss_elimination_with_partial_pivoting(A, b)
end = time.time()
display("Rozwiązanie:", x)
print(f"Czas wykonania: {end - start:.6f} sekund")
```
```
[54] ✓ 0.0s
... 'Rozwiązanie:'
... array([-1.6949,  1.8708,  0.7768,  1.0932, -0.3524,  2.0929,  1.1428, -0.332 ,  0.3586, -1.5577, -1.0116, -1.2818,  0.2741])
... Czas wykonania: 0.001004 sekund
```

Rys. 1.2. wyniki

1.9. Algorytm LU faktoryzacji bez pivotingu

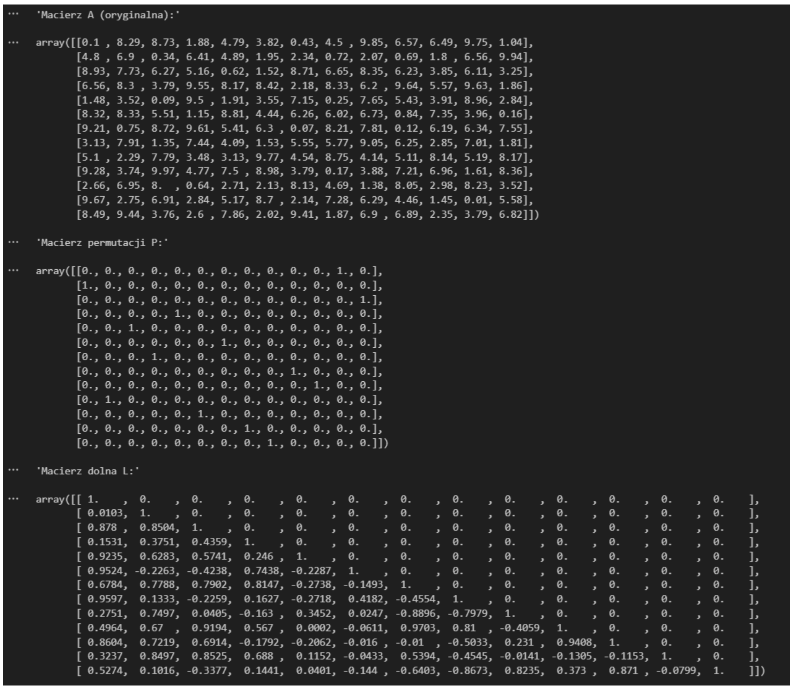
```
···  'Macierz L (dolna trójkątna):'

···  array([[ 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [48.    , 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [89.3   , 1.8735, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [65.6   , 1.3696, 0.4084, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [14.8   , 0.3048, -0.1356, 1.9286, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [83.2   , 1.7426, 0.7931, -1.3997, -1.3465, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [92.1   , 1.9507, 1.9306, 3.2668, 0.0481, 6.9544, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [31.3   , 0.6434, -0.2272, 0.3705, 0.41  , -0.2174, -0.0399, 1.    , 0.    , 0.    , 0.    , 0.    , 0.    ],
            [51.    , 1.0754, 1.155 , 1.2843, -0.1643, -0.1386, -0.0508, -0.8728, 1.    , 0.    , 0.    , 0.    , 0.    ],
            [92.8   , 1.9579, 1.7641, 1.326 , -0.7793, 3.7759, 0.4566, -1.8888, 0.3245, 1.    , 0.    , 0.    , 0.    ],
            [26.6   , 0.5462, 0.402 , -0.3893, -0.2221, 0.1336, -0.0589, 0.1192, 3.4914, 2.4117, 1.    , 0.    , 0.    ],
            [96.7   , 2.0431, 1.6356, 0.4755, -0.9033, 1.2079, 0.1293, -1.1464, -1.468 , -0.9863, -2.3637, 1.    , 0.    ],
            [84.9   , 1.7758, 0.5509, -1.4999, -1.0848, 0.827 , -0.0307, -0.0431, 0.0638, 0.8541, -0.404 , 0.3406, 1.    ]])

···  'Macierz U (górna trójkątna):'

···  array([[   0.1   ,    8.29  ,    8.73  ,    1.88  ,    4.79  ,    3.82  ,    0.43  ,    4.5   ,    9.85  ,    6.57  ,    6.49  ,    9.75  ,    1.04  ],
            [   0.    , -391.02  , -418.7   ,  -83.83  , -225.03  , -181.41  ,  -18.3   , -215.28  , -470.73  , -314.67  , -309.72  , -461.44  ,  -39.98  ],
            [   0.    ,    0.    ,  11.1058,   -5.6704,   -5.5385,    0.2615,    4.5956,    8.1221,   10.6469,    9.056 ,    4.5463,   -0.0677,  -14.7204],
            [   0.    ,    0.    ,    0.    ,    3.3476,    4.3991,    6.1725,   -2.8418,    4.6513,    0.3835,    5.9081,    2.1485,    2.0258,   -5.5977],
            [   0.    ,    0.    ,    0.    ,    0.    ,   -9.6342,   -9.5661,   12.4673,   -8.6078,    6.0395,   -6.0699,   -1.2751,    1.3778,    8.4326],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,   -1.7048,   11.539 ,   -4.7512,    7.7373,   -4.5206,    4.79  ,    1.6175,   -1.5043],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,  -84.2708,   16.2849,  -57.0333,    3.7926,  -36.4171,   -9.3107,   46.5165],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    6.6905,    2.8044,    4.5813,   -0.6755,   -2.6429,   -8.222 ],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,   -3.1635,   -7.0458,    0.2262,   -0.6834,   18.6773],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,   11.338 ,   -3.5867,   -7.8631,   -7.0623],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    3.3894,   22.928 ,  -40.9554],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,   42.0023,  -68.9246],
            [   0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,    0.    ,   12.4689]])
```

Rys. 1.3. wyniki

```
···  'Rozwiązanie z numpy.linalg.solve (kontrola):'     'Rozwiązanie x:'       'Sprawdzenie: L @ U @ x ='    'Oryginalny wektor D:'

···  array([[-1.6949],                                  array([[-1.6949],      array([[4.],                  array([[4],
            [ 1.8708],                                        [ 1.8708],             [9.],                        [9],
            [ 0.7768],                                        [ 0.7768],             [3.],                        [3],
            [ 1.0932],                                        [ 1.0932],             [2.],                        [2],
            [-0.3524],                                        [-0.3524],             [9.],                        [9],
            [ 2.0929],                                        [ 2.0929],             [7.],                        [7],
            [ 1.1428],                                        [ 1.1428],             [2.],                        [7],
            [-0.332 ],                                        [-0.332 ],             [7.],                        [7],
            [ 0.3586],                                        [ 0.3586],             [8.],                        [8],
            [-1.5577],                                        [-1.5577],             [8.],                        |8|,
            [-1.0116],                                        [-1.0116],             [2.],                        [2],
            [-1.2818],                                        [-1.2818],             [9.],                        [9],
            [ 0.2741]])                                       [ 0.2741]])            [7.]])                       [7]])

                                                                                                          Czas wykonania: 0.011038 sekund
```

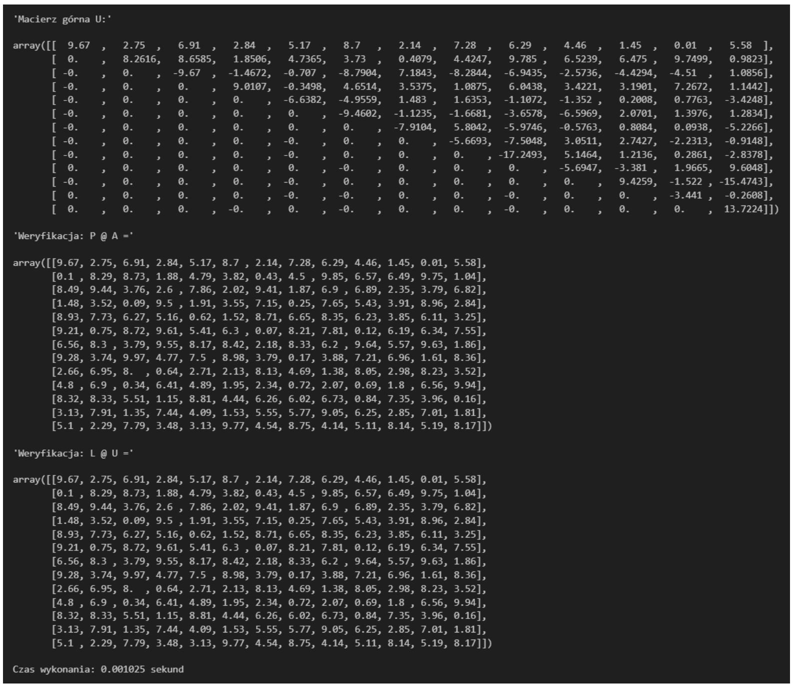
Rys. 1.4. wyniki

1.10. Algorytm LU faktoryzacji z pivotingiem



```
···  'Macierz A (oryginalna):'

···  array([[0.1 , 8.29, 8.73, 1.88, 4.79, 3.82, 0.43, 4.5 , 9.85, 6.57, 6.49, 9.75, 1.04],
            [4.8 , 6.9 , 0.34, 6.41, 4.89, 1.95, 2.34, 0.72, 2.07, 0.69, 1.8 , 6.56, 9.94],
            [8.93, 7.73, 6.27, 5.16, 0.62, 1.52, 8.71, 6.65, 8.35, 6.23, 3.85, 6.11, 3.25],
            [6.56, 8.3 , 3.79, 9.55, 8.17, 8.42, 2.18, 8.33, 6.2 , 9.64, 5.57, 9.63, 1.86],
            [1.48, 3.52, 0.09, 9.5 , 1.91, 3.55, 7.15, 0.25, 7.65, 5.43, 3.91, 8.96, 2.84],
            [8.32, 8.33, 5.51, 1.15, 8.81, 4.44, 6.26, 6.02, 6.73, 0.84, 7.35, 3.96, 0.16],
            [9.21, 0.75, 8.72, 9.61, 5.41, 6.3 , 0.07, 8.21, 7.81, 0.12, 6.19, 6.34, 7.55],
            [3.13, 7.91, 1.35, 7.44, 4.09, 1.53, 5.55, 5.77, 9.05, 6.25, 2.85, 7.01, 1.81],
            [5.1 , 2.29, 7.79, 3.48, 3.13, 9.77, 4.54, 8.75, 4.14, 5.11, 8.14, 5.19, 8.17],
            [9.28, 3.74, 9.97, 4.77, 7.5 , 8.98, 3.79, 0.17, 3.88, 7.21, 6.96, 1.61, 8.36],
            [2.66, 6.95, 8.  , 0.64, 2.71, 2.13, 8.13, 4.69, 1.38, 8.05, 2.98, 8.23, 3.52],
            [9.67, 2.75, 6.91, 2.84, 5.17, 8.7 , 2.14, 7.28, 6.29, 4.46, 1.45, 0.01, 5.58],
            [8.49, 9.44, 3.76, 2.6 , 7.86, 2.02, 9.41, 1.87, 6.9 , 6.89, 2.35, 3.79, 6.82]])

···  'Macierz permutacji P:'

···  array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
            [1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
            [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
            [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]])

···  'Macierz dolna L:'

···  array([[ 1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.0103,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.878 ,  0.8504,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.1531,  0.3751,  0.4359,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.9235,  0.6283,  0.5741,  0.246 ,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.9524, -0.2263, -0.4238,  0.7438, -0.2287,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.6784,  0.7788,  0.7902,  0.8147, -0.2738, -0.1493,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.9597,  0.1333, -0.2259,  0.1627, -0.2718,  0.4182, -0.4554,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.2751,  0.7497,  0.0405, -0.163 ,  0.3452,  0.0247, -0.8896, -0.7979,  1.    ,  0.    ,  0.    ,  0.    ,  0.    ],
            [ 0.4964,  0.67  ,  0.9194,  0.567 ,  0.0002, -0.0611,  0.9703,  0.81  , -0.4059,  1.    ,  0.    ,  0.    ,  0.    ],
            [ 0.8604,  0.7219,  0.6914, -0.1792, -0.2062, -0.016 , -0.01  , -0.5033,  0.231 ,  0.9408,  1.    ,  0.    ,  0.    ],
            [ 0.3237,  0.8497,  0.8525,  0.688 ,  0.1152, -0.0433,  0.5394, -0.4545, -0.0141, -0.1305, -0.1153,  1.    ,  0.    ],
            [ 0.5274,  0.1016, -0.3377,  0.1441,  0.0401, -0.144 , -0.6403, -0.8673,  0.8235,  0.373 ,  0.871 , -0.0799,  1.    ]])
```

Rys. 1.5. wyniki



```
'Macierz górna U:'

array([[  9.67  ,   2.75  ,   6.91  ,   2.84  ,   5.17  ,   8.7   ,   2.14  ,   7.28  ,   6.29  ,   4.46  ,   1.45  ,   0.01  ,   5.58  ],
       [  0.    ,   8.2616,   8.6585,   1.8506,   4.7365,   3.73  ,   0.4079,   4.4247,   9.785 ,   6.5239,   6.475 ,   9.7499,   0.9823],
       [ -0.    ,   0.    ,  -9.67  ,  -1.4672,  -0.707 ,  -8.7904,   7.1843,  -8.2844,  -6.9435,  -2.5736,  -4.4294,  -4.51  ,   1.0856],
       [ -0.    ,   0.    ,   0.    ,   9.0107,  -0.3498,   4.6514,   3.5375,   1.0875,   6.0438,   3.4221,   3.1901,   7.2672,   1.1442],
       [ -0.    ,   0.    ,   0.    ,   0.    ,  -6.6382,  -4.9559,   1.483 ,   1.6353,  -1.1072,  -1.352 ,   0.2008,   0.7763,  -3.4248],
       [ -0.    ,   0.    ,   0.    ,   0.    ,   0.    ,  -9.4602,  -1.1235,  -1.6681,  -3.6578,  -6.5969,   2.0701,   1.3976,   1.2834],
       [ -0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,  -7.9104,   5.8042,  -5.9746,  -0.5763,   0.8084,   0.0938,  -5.2266],
       [ -0.    ,   0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,  -5.6693,  -7.5048,   3.0511,   2.7427,  -2.2313,  -0.9148],
       [ -0.    ,   0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,   0.    , -17.2493,   5.1464,   1.2136,   0.2861,  -2.8378],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,   0.    ,  -5.6947,  -3.381 ,   1.9665,   9.6048],
       [ -0.    ,   0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   9.4259,  -1.522 , -15.4743],
       [  0.    ,   0.    ,   0.    ,   0.    ,  -0.    ,  -0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,  -3.441 ,  -0.2608],
       [  0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,  -0.    ,   0.    ,   0.    ,   0.    ,  13.7224]])

'Weryfikacja: P @ A ='

array([[9.67, 2.75, 6.91, 2.84, 5.17, 8.7 , 2.14, 7.28, 6.29, 4.46, 1.45, 0.01, 5.58],
       [0.1 , 8.29, 8.73, 1.88, 4.79, 3.82, 0.43, 4.5 , 9.85, 6.57, 6.49, 9.75, 1.04],
       [8.49, 9.44, 3.76, 2.6 , 7.86, 2.02, 9.41, 1.87, 6.9 , 6.89, 2.35, 3.79, 6.82],
       [1.48, 3.52, 0.09, 9.5 , 1.91, 3.55, 7.15, 0.25, 7.65, 5.43, 3.91, 8.96, 2.84],
       [8.93, 7.73, 6.27, 5.16, 0.62, 1.52, 8.71, 6.65, 8.35, 6.23, 3.85, 6.11, 3.25],
       [9.21, 0.75, 8.72, 9.61, 5.41, 6.3 , 0.07, 8.21, 7.81, 0.12, 6.19, 6.34, 7.55],
       [6.56, 8.3 , 3.79, 9.55, 8.17, 8.42, 2.18, 8.33, 6.2 , 9.64, 5.57, 9.63, 1.86],
       [9.28, 3.74, 9.97, 4.77, 7.5 , 8.98, 3.79, 0.17, 3.88, 7.21, 6.96, 1.61, 8.36],
       [2.66, 6.95, 8.  , 0.64, 2.71, 2.13, 8.13, 4.69, 1.38, 8.05, 2.98, 8.23, 3.52],
       [4.8 , 6.9 , 0.34, 6.41, 4.89, 1.95, 2.34, 0.72, 2.07, 0.69, 1.8 , 6.56, 9.94],
       [8.32, 8.33, 5.51, 1.15, 8.81, 4.44, 6.26, 6.02, 6.73, 0.84, 7.35, 3.96, 0.16],
       [3.13, 7.91, 1.35, 7.44, 4.09, 1.53, 5.55, 5.77, 9.05, 6.25, 2.85, 7.01, 1.81],
       [5.1 , 2.29, 7.79, 3.48, 3.13, 9.77, 4.54, 8.75, 4.14, 5.11, 8.14, 5.19, 8.17]])

'Weryfikacja: L @ U ='

array([[9.67, 2.75, 6.91, 2.84, 5.17, 8.7 , 2.14, 7.28, 6.29, 4.46, 1.45, 0.01, 5.58],
       [0.1 , 8.29, 8.73, 1.88, 4.79, 3.82, 0.43, 4.5 , 9.85, 6.57, 6.49, 9.75, 1.04],
       [8.49, 9.44, 3.76, 2.6 , 7.86, 2.02, 9.41, 1.87, 6.9 , 6.89, 2.35, 3.79, 6.82],
       [1.48, 3.52, 0.09, 9.5 , 1.91, 3.55, 7.15, 0.25, 7.65, 5.43, 3.91, 8.96, 2.84],
       [8.93, 7.73, 6.27, 5.16, 0.62, 1.52, 8.71, 6.65, 8.35, 6.23, 3.85, 6.11, 3.25],
       [9.21, 0.75, 8.72, 9.61, 5.41, 6.3 , 0.07, 8.21, 7.81, 0.12, 6.19, 6.34, 7.55],
       [6.56, 8.3 , 3.79, 9.55, 8.17, 8.42, 2.18, 8.33, 6.2 , 9.64, 5.57, 9.63, 1.86],
       [9.28, 3.74, 9.97, 4.77, 7.5 , 8.98, 3.79, 0.17, 3.88, 7.21, 6.96, 1.61, 8.36],
       [2.66, 6.95, 8.  , 0.64, 2.71, 2.13, 8.13, 4.69, 1.38, 8.05, 2.98, 8.23, 3.52],
       [4.8 , 6.9 , 0.34, 6.41, 4.89, 1.95, 2.34, 0.72, 2.07, 0.69, 1.8 , 6.56, 9.94],
       [8.32, 8.33, 5.51, 1.15, 8.81, 4.44, 6.26, 6.02, 6.73, 0.84, 7.35, 3.96, 0.16],
       [3.13, 7.91, 1.35, 7.44, 4.09, 1.53, 5.55, 5.77, 9.05, 6.25, 2.85, 7.01, 1.81],
       [5.1 , 2.29, 7.79, 3.48, 3.13, 9.77, 4.54, 8.75, 4.14, 5.11, 8.14, 5.19, 8.17]])

Czas wykonania: 0.001025 sekund
```

Rys. 1.6. wyniki

1.11. porównanie czsów obliczeń

Metoda	**Czas wykonania [s]**
Gauss bez pivotingu	0.004109
Gauss z pivotingiem	0.001004
LU bez pivotingu	0.011038
LU z pivotingiem	0.001025

Tabela 1.1. Porównanie czasów wykonania różnych metod rozwiązywania układów równań

1.12. Algorytm Gaussa vs LU bez pivotingu przy stałym A i różnych b

Metoda	**Suma czasów [s]**
Gauss bez pivotingu	0.003426
LU bez pivotingu	0.001007

Tabela 1.2. Porównanie czasu działania metody Gaussa i LU bez pivotingu dla stałego A i różnych wektorów b

1.13. Wnioski

Pivoting przyśpiesza obliczenia. LU jest szczególnie użyteczne podczas wielokrotnego obliczania układu równań, w którym macierz A jest stała, a wektor b się zmienia. Ponieważ złożoność obliczeniowa jest redukowana do $O(n^2)$ dla kolejnych rozwiązań układu, dla nowych wektorów b.