Akademia Górniczo-Hutnicza

Wydział Informatyki

Algorytmy macierzowe

Ćwiczenie 4

Dodawanie i mnożenie macierzy hierarchicznych

Jakub Szewczyk i Albert Arnautov

Spis treści

1 Wstęp

Celem ćwiczenia było zaimplementowanie algorytmu mnożenia i dodawania macierzy skompresowanych algorytmem hierarchicznej kompresji z ćwiczenia 3.

2 Pseudokod

2.1 Algorytm mnożenia macierzy skompresowanej przez wektor

Algorytm 1 Mnożenie macierzy hierarchicznej przez wektor

Require: Węzeł macierzy v, wektor x
Ensure: Wektor $y = v \cdot x$

if $v.children = \emptyset$ **then** ▷ liść
  if $v.rank = 0$ **then**
    return wektor zerowy odpowiedniego rozmiaru
  end if
  return $v.U \cdot (\text{diag}(v.\text{singular\_vals}) \cdot (v.V \cdot x))$
end if
$n \leftarrow \text{length}(x)$
$m \leftarrow n/2$
$x_1 \leftarrow x[1 : m]$
$x_2 \leftarrow x[m + 1 : n]$
$y_1 \leftarrow \text{mat\_vec}(v.children[0], x_1) + \text{mat\_vec}(v.children[1], x_2)$
$y_2 \leftarrow \text{mat\_vec}(v.children[2], x_1) + \text{mat\_vec}(v.children[3], x_2)$
return $\text{concatenate}(y_1, y_2)$

2.2 Algorytm mnożenia macierzy skompresowanych

Algorytm 2 Hierarchiczne mnożenie macierzy

Require: nodes v, w
Ensure: $Y = v \cdot w$

if $v.sons = 0$ **and** $w.sons = 0$ **then**
  if $v.rank = 0$ **and** $w.rank = 0$ **then**
    return zero matrix of appropriate size
  else
    return $v.U \cdot (v.V \cdot w.U) \cdot w.V$
  end if
end if
if $v.sons = 0$ **and** $w.sons > 0$ **then**
  $A = U_v V_v$
  Podziel $U_v = [U'_v\ U''_v]$
  Podziel $V_v = \begin{bmatrix} V'_v \\ V''_v \end{bmatrix}$
  return

$$\begin{bmatrix} \text{mult}(U'_v V'_v, w.sons[0]) + \text{mult}(U'_v V''_v, w.sons[2]) & \text{mult}(U'_v V'_v, w.sons[1]) + \text{mult}(U'_v V''_v, w.sons[3]) \\ \text{mult}(U''_v V'_v, w.sons[0]) + \text{mult}(U''_v V''_v, w.sons[2]) & \text{mult}(U''_v V'_v, w.sons[1]) + \text{mult}(U''_v V''_v, w.sons[3]) \end{bmatrix}$$

end if
if $v.sons > 0$ **and** $w.sons = 0$ **then**
  $B = U_w V_w$
  Podziel $U_w = [U'_w\ U''_w]$
  Podziel $V_w = \begin{bmatrix} V'_w \\ V''_w \end{bmatrix}$
  return

$$\begin{bmatrix} \text{mult}(v.sons[0], U'_w V'_w) + \text{mult}(v.sons[1], U''_w V'_w) & \text{mult}(v.sons[0], U'_w V''_w) + \text{mult}(v.sons[1], U''_w V''_w) \\ \text{mult}(v.sons[2], U'_w V'_w) + \text{mult}(v.sons[3], U''_w V'_w) & \text{mult}(v.sons[2], U'_w V''_w) + \text{mult}(v.sons[3], U''_w V''_w) \end{bmatrix}$$

end if
if $v.sons > 0$ **and** $w.sons > 0$ **then**
  return

$$\begin{bmatrix} \text{mult}(v.sons[0], w.sons[0]) + \text{mult}(v.sons[1], w.sons[2]) & \text{mult}(v.sons[0], w.sons[1]) + \text{mult}(v.sons[1], w.sons[3]) \\ \text{mult}(v.sons[2], w.sons[0]) + \text{mult}(v.sons[3], w.sons[2]) & \text{mult}(v.sons[2], w.sons[1]) + \text{mult}(v.sons[3], w.sons[3]) \end{bmatrix}$$

end if
if v i w są liczbami **then**
  return $v \cdot w$
end if

2.3 Algorytm dodawania macierzy skompresowanych

Algorytm 3 Hierarchiczne dodawanie macierzy (pełna wersja)

Require: węzły v, w

if $v.sons = 0$ **and** $w.sons = 0$ **and** $v.rank = 0$ **and** $w.rank = 0$ **then**
  return zero matrix of proper dimensions
end if
if $v.sons = 0$ **and** $w.sons = 0$ **and** $v.rank \neq 0$ **and** $w.rank \neq 0$ **then**
  return rSVDofCompressed([v.U, w.U], [v.V, w.V])
end if
if $v.sons = 0$ **and** $w.sons > 0$ **then**
  $A = U_v V_v$
  Podziel $U_v = [U'_v \; U''_v]$
  Podziel $V_v = \begin{bmatrix} V'_v \\ V''_v \end{bmatrix}$
  return

$$\begin{bmatrix} \text{add}(U'_v V'_v, w.sons[0]) & \text{add}(U'_v V''_v, w.sons[1]) \\ \text{add}(U''_v V'_v, w.sons[2]) & \text{add}(U''_v V''_v, w.sons[3]) \end{bmatrix}$$

end if
if $v.sons > 0$ **and** $w.sons = 0$ **then**
  $B = U_w V_w$
  Podziel $U_w = [U'_w \; U''_w]$
  Podziel $V_w = \begin{bmatrix} V'_w \\ V''_w \end{bmatrix}$
  return

$$\begin{bmatrix} \text{add}(v.sons[0], U'_w V'_w) & \text{add}(v.sons[1], U'_w V''_w) \\ \text{add}(v.sons[2], U''_w V'_w) & \text{add}(v.sons[3], U''_w V''_w) \end{bmatrix}$$

end if
if $v.sons > 0$ **and** $w.sons > 0$ **then**
  return

$$\begin{bmatrix} \text{add}(v.sons[0], w.sons[0]) & \text{add}(v.sons[1], w.sons[1]) \\ \text{add}(v.sons[2], w.sons[2]) & \text{add}(v.sons[3], w.sons[3]) \end{bmatrix}$$

end if
if v i w są liczbami **then**
  return $v + w$
end if

3 Ważne fragmenty kodu

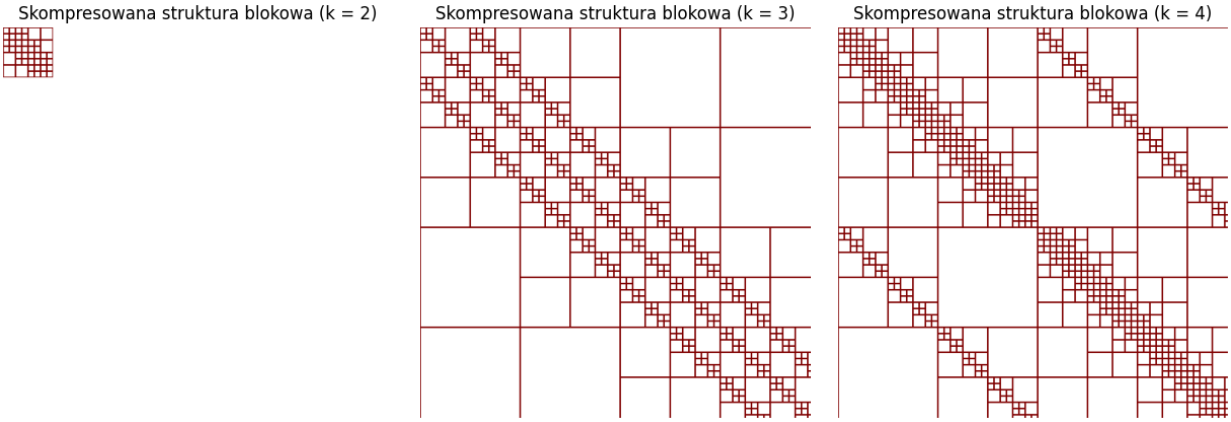
3.1 Mnożenie macierzy

```
def matrix_matrix_mult(v, w, target_rank):

    if isinstance(v, MatrixLeaf) and isinstance(w, MatrixLeaf):

        if v.rank == 0 or w.rank == 0:
            return create_zero_leaf(v.size)


        V_v_scaled = np.diag(v.singular_vals) @ v.V
        U_w_scaled = w.U @ np.diag(w.singular_vals)
        middle = V_v_scaled @ U_w_scaled

        U_new = v.U @ middle
        V_new = w.V

        if U_new.size > 0 and V_new.size > 0:
            A_temp = U_new @ V_new
```

```
18
19              if np.all(np.abs(A_temp) < eps):
20                  return create_zero_leaf(v.size)
21
22              U_svd, D_svd, V_svd = svd(A_temp)
23
24              k = min(target_rank, len(D_svd), len([d for d in D_svd if d > eps]))
25
26              if k == 0:
27                  return create_zero_leaf(v.size)
28
29              A_result = U_svd[:, :k] @ np.diag(D_svd[:k]) @ V_svd[:k, :]
30              return MatrixLeaf(A_result, v.size, U_svd, D_svd, V_svd, k)
31
32          return create_zero_leaf(v.size)
33
34      if isinstance(v, MatrixNode) and isinstance(w, MatrixNode):
35
36          result_node = MatrixNode()
37
38          prod1 = matrix_matrix_mult(v.children[0], w.children[0], target_rank)
39          prod2 = matrix_matrix_mult(v.children[1], w.children[2], target_rank)
40          c00 = matrix_matrix_add(prod1, prod2, target_rank)
41          result_node.children.append(c00)
42
43          prod3 = matrix_matrix_mult(v.children[0], w.children[1], target_rank)
44          prod4 = matrix_matrix_mult(v.children[1], w.children[3], target_rank)
45          c01 = matrix_matrix_add(prod3, prod4, target_rank)
46          result_node.children.append(c01)
47
48          prod5 = matrix_matrix_mult(v.children[2], w.children[0], target_rank)
49          prod6 = matrix_matrix_mult(v.children[3], w.children[2], target_rank)
50          c10 = matrix_matrix_add(prod5, prod6, target_rank)
51          result_node.children.append(c10)
52
53          prod7 = matrix_matrix_mult(v.children[2], w.children[1], target_rank)
54          prod8 = matrix_matrix_mult(v.children[3], w.children[3], target_rank)
55          c11 = matrix_matrix_add(prod7, prod8, target_rank)
56          result_node.children.append(c11)
57
58          return result_node
59
60      return v
```

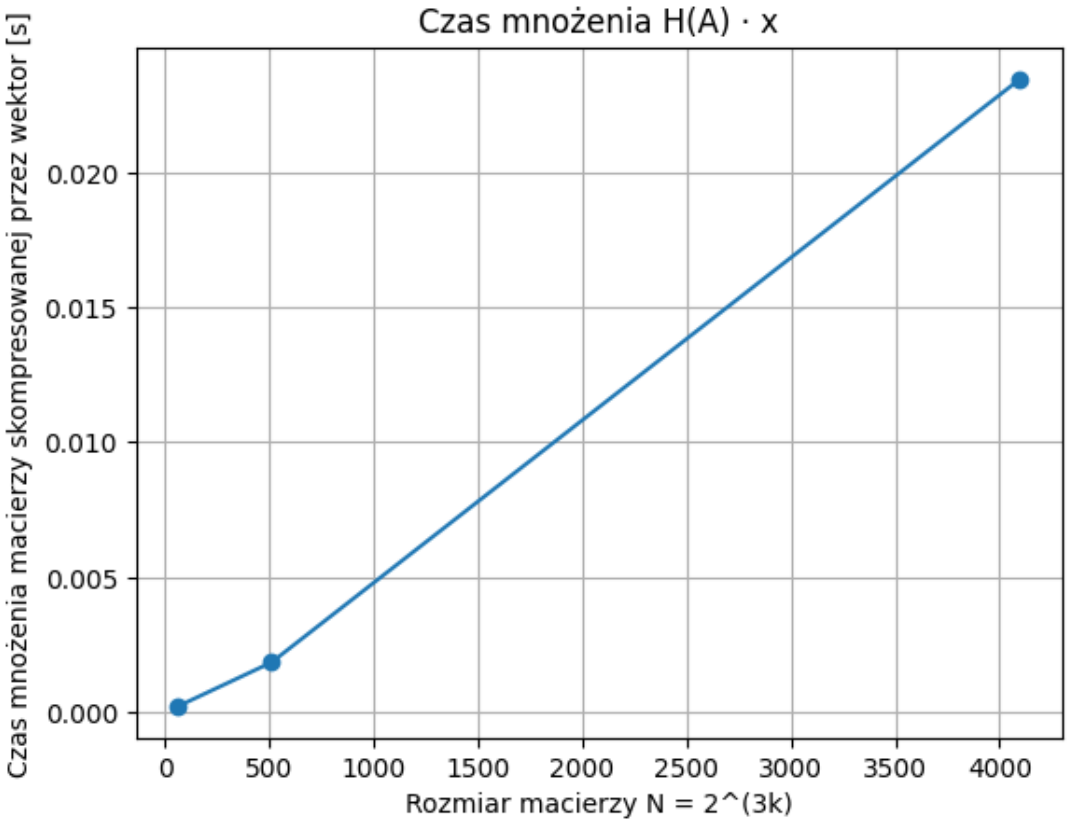
4 Wyniki

4.1 Bitmapa skompresowanych macierzy dla $k = \{2, 3, 4\}$

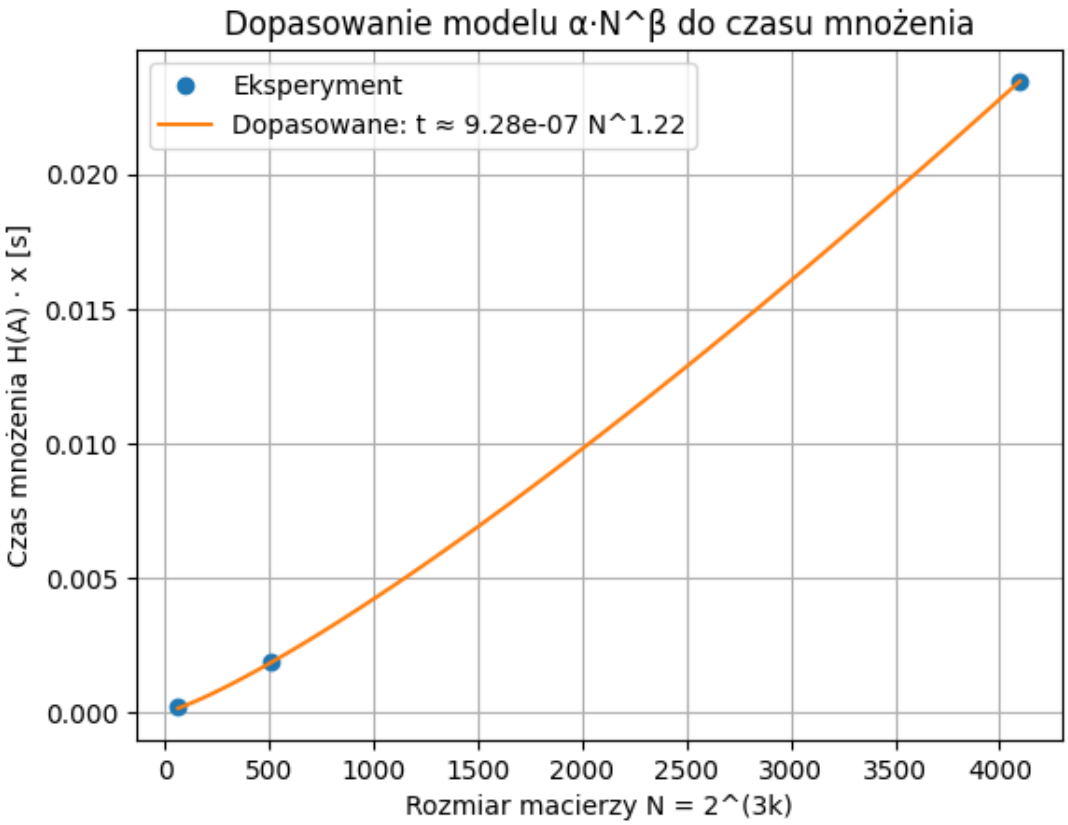


Rysunek 1: Skompresowane macierze dla poszczególnych k

4.2 Wykres czasów mnożenia macierzy przez wektor



Rysunek 2: Czas mnożenia macierzy przez wektor

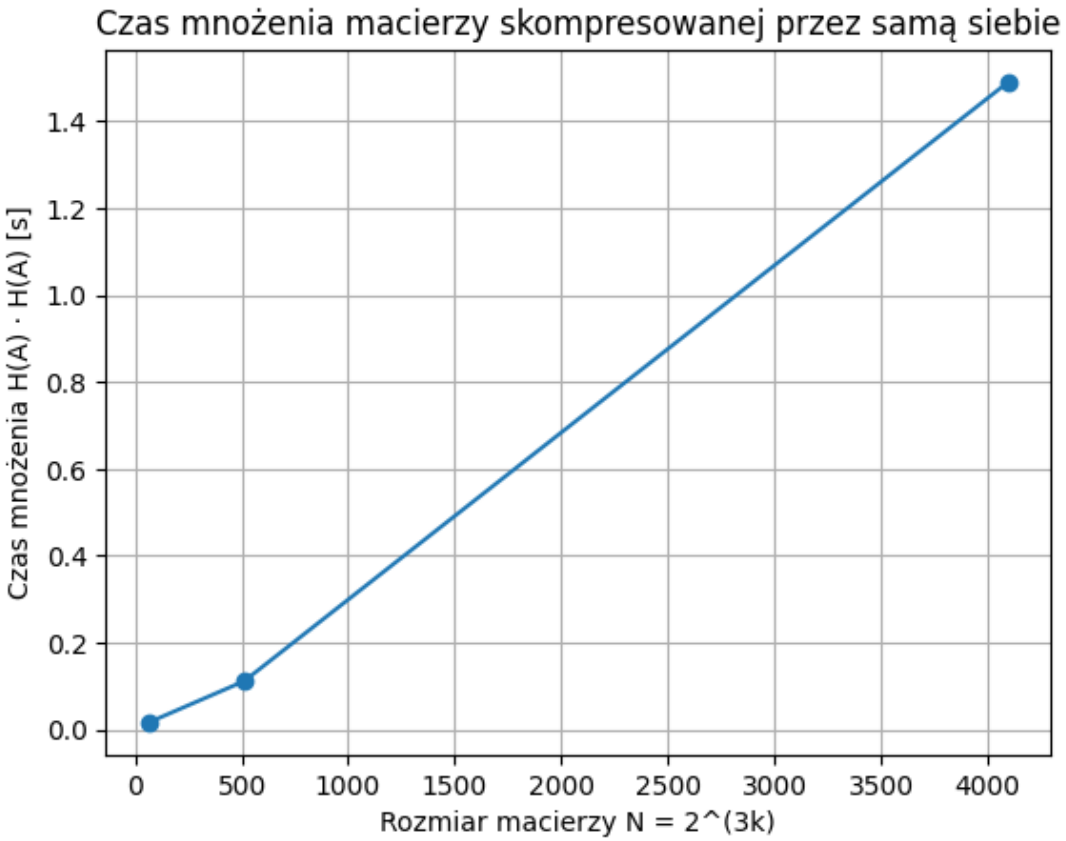


Rysunek 3: Dopasowanie wykresu do czasu mnożenia

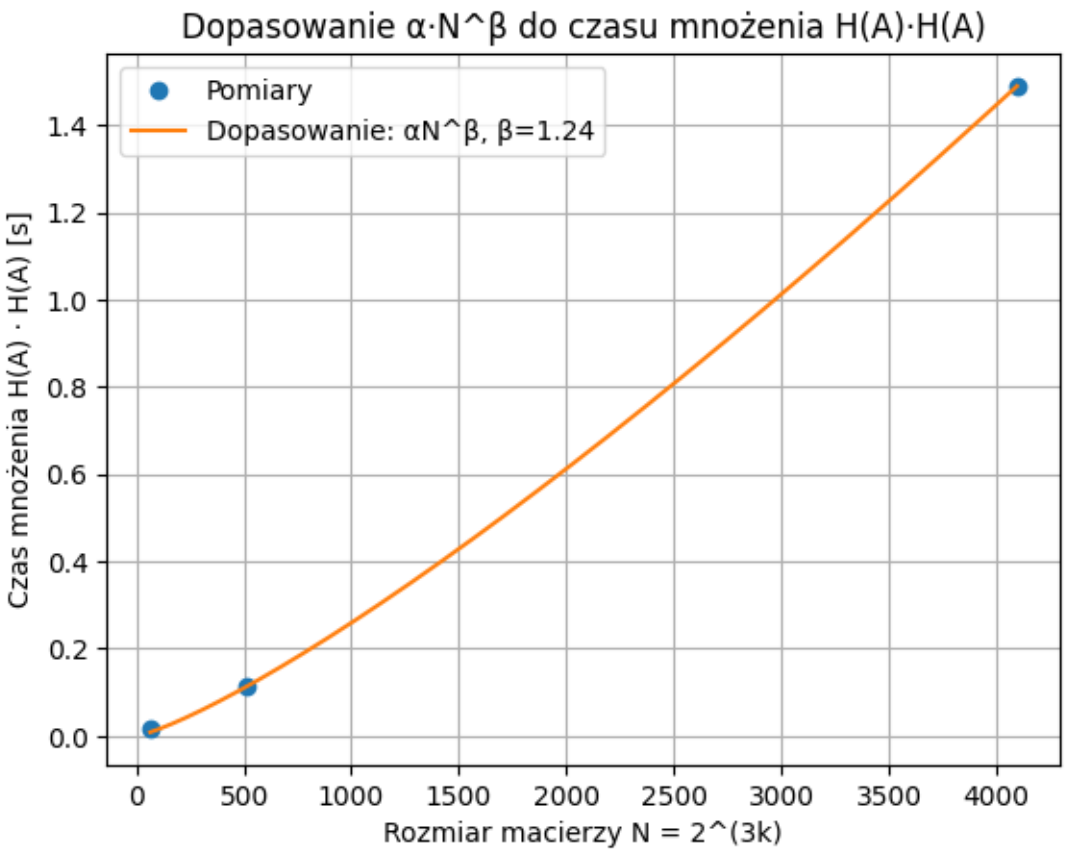
Porównanie wyników mnożenia macierzy gęstej z wektorem po rekonstrukcji:

$$\|Ax - Hx\|_2^2 = \sum_i (Ax_i - Hx_i)^2 = 1.15 \cdot 10^{-28}$$

4.3 Wykres czasów mnożenia macierzy przez samą siebie



Rysunek 4: Czas mnożenia macierzy przez samą siebie



Rysunek 5: Dopasowanie prostej do czasu mnożenia

Porównanie wyników mnożenia macierzy gęstej z macierzą po rekonstrukcji:

$$\|Ax - Hx\|_2^2 = \sum_i (Ax_i - Hx_i)^2 = 3.55 \cdot 10^2$$

4.4 Wyniki dodawania macierzy

Do sprawdzenia poprawności kodu użyta została następująca macierz U i wektor **x**:

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

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

Poniżej przedstawiono wynik mnożenia skompresowanej macierzy U przez wektor **x** oraz porównanie z wynikiem otrzymanym mnożeniem przez macierz nieskompresowaną:

```
U = create_U_matrix()
r = 1
epsilon = 0
compressed_tree = CompressMatrix(U, r, epsilon)
```
[ ]

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8], dtype=float)
```
[15] ✓ 0.0s

```
y_normal = U @ x
y_compressed = matrix_vector_mult(compressed_tree, x)
print("Wynik klasyczny:")
print(y_normal)
print("Wynik skompresowany:")
print(y_compressed)
```
[16] ✓ 0.0s

```
Wynik klasyczny:
[ 3.  3.  7.  7. 11. 11. 15. 15.]
Wynik skompresowany:
[ 3.  3.  7.  7. 11. 11. 15. 15.]
```

Rysunek 6: Sprawdzenie mnożenia macierzy podanej na zajęciach