Algorytmy macierzowe

Laboratorium 4
Sprawozdanie

Algorytmy permutacji macierzy rzadkich

Adam Naumiec i Andrzej Zaborniak

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki
Grudzień MMXXIII

Algorytmy macierzowe
Laboratorium 4

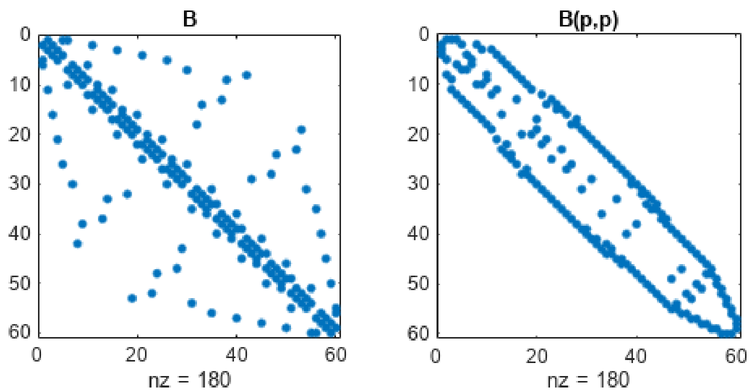
Adam Naumiec Andrzej Zaborniak

Grudzień 2023

Spis treści

0 Abstrakt

Niniejszy dokument jest sprawozdaniem z wykonania Laboratorium 4 z przedmiotu Algorytmy macierzowe prowadzonego przez Pana prof. dr hab. Macieja Paszyńskiego w roku akademickim 2023/2024 na piątym semestrze studiów pierwszego stopnia na kierunku Informatyka prowadzonego na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie na Wydziale Informatyki.



Rysunek 1: Wizualizacja wyniku działania algorytmu Reversed Cuthill-Mckee (2)

1 Streszczenie wykładu

Czwarty wykład skupiony był wokół permutacji macierzy rzadkich. Poruszane były tematy:

- permutacja macierzy i wykorzystanie eliminacji Gaussa
- macierz symetryczna i graf eliminacji
- proces eliminacji na grafie nieskierowanym
- proces eliminacji na grafie skierowanym
- fill-in (wypełnianie)
- algorytm minimum degree
- minimum degree na grafie eliminacji
- reordering
- nieoptymalność minimum degree
- algorytm Cuthill-McKnee
- ordering BFS na grafie eliminacji
- algorytm reversed Cuthill-McKnee

Zaprezentowane zostało także wykorzystanie i znaczenie tych algorytmów w praktyce oraz historia wykorzystania, postrzegania i zastosowania macierzy, a także matematyczne, inofmratyczne i algorytmiczne podstawy operacji na macierzach.

2 Zadania zrealizowane w ramach laboratorium i raport z laboratorium

W ramach laboratorium zrealizowano zadanie polegające na implementacji, porównaniu wyników i przygotowaniu sprawozdania wraz z wnioskami z laboratorium

2.1 Zadania

1. Proszę wybrać ulubiony język programowania.
2. Dla $k = 2, 3, 4$
   - Proszę wygenerować macierz o rozmiarze $2^{3k} = 2^k * 2^k * 2^k$ o strukturze opisującej topologię trójwymiarowej siatki zbudowanej z elementów sześciennych (wiersz = wierzchołek, niezerowe losowe wartości w kolumnach – sąsiadujące wierzchołki siatki)
   - Proszę dokonać kompresji macierzy do macierzy hierarchicznej
   - Proszę uruchomić algorytmy kompresji macierzy rzadkiej
   - Proszę napisać algorytmy permutacji macierzy
     - Minimum degree
     - Cuthill-McKee
     - Reversed Cuthill-McKee [odwrócona permutacja Cuthill-McKee (fajny algorytm)]
   - Proszę porównać stopień kompresji macierzy przed i po permutacji macierzy

2.2 Raporty

1. Proszę przedstawić pseudo-kod algorytmu permutacji macierzy
2. Proszę narysować wzorzec rzadkości (sparsity patern) macierzy rzadkiej przed kompresją i permutacją (w MATLABie spy(A))
3. Proszę narysować macierz rzadką przed permutacją ale po kompresji (używając rysowacza macierzy hierarchicznych)
4. Proszę narysować wzorzec rzadkości macierzy rzadkiej po permutacji ale przed kompresją
5. Proszę narysować macierz rzadką po permutacji i po kompresji (używając rysowacza macierzy hierarchicznych)
6. Proszę napisać i opisać kod algorytmu permutacji macierzy

3 Pseudokody algorytmów i fragmenty kodu oraz opis algorytmów permutacji macierzy

PRZYGOTOWANO pseudokody testowancyh algorytmów oraz przedstawiono wybrane najważniejsze fragmenty kodu zaimplementowanych algorytmów w języku programowania wysokiego poziomu - wybrano język Python w wersji 3.11 oraz potrzebne biblioteki.

3.1 Opisy algorytmów permutacji macierzy

3.1.1 Algorytm permutacji macierzy Minimum Degree

Algorytm minimalnego stopnia to technika stosowana do permutacji macierzy w celu zmniejszenia złożoności obliczeniowej i pamięciowej podczas rozwiązywania układów równań liniowych. Jest często stosowany w kontekście macierzy rzadkich.

Podstawowy pomysł algorytmu minimalnego stopnia polega na wyborze wierzchołka o minimalnym stopniu w grafie macierzy i eliminacji tego wierzchołka (co odpowiada eliminacji Gaussa dla danego wiersza i kolumny w macierzy). Stopień wierzchołka to liczba jego sąsiadów. Eliminacja wierzchołka powoduje, że jego sąsiedzi stają się połączeni, co zwiększa ich stopień. Proces jest powtarzany, aż wszystkie wierzchołki zostaną wyeliminowane. Kolejność eliminacji wierzchołków daje permutację macierzy.

Algorithm 1 Algorytm permutacji Minimum Degree

$G \leftarrow (V, E)$ ▷ Graf eliminacji, gdzie V to zbiór wierzchołków, E to zbiór krawędzi
for each vertex $v_i \in V$ **do**
    $t_i \leftarrow$ liczba sąsiadów v_i w $G_0 = (V, E)$
end for
for $k = 1$ to n **do** ▷ n to liczba wierzchołków w grafie
    Wybierz wierzchołek p o minimalnej wartości t_i spośród wierzchołków w $V_{G_{k-1}}$
    for each vertex v_i sąsiadujący z p w G_{k-1} **do**
        Zaktualizuj zbiór sąsiadów v_i przez dodanie sąsiadów p i usunięcie v_i i p
        Zaktualizuj t_i jako liczbę sąsiadów v_i w G_k
    end for
    Usuń p z V_k
end for

3.1.2 Algorytm permutacji macierzy Cuthill-McKee

Algorytm Cuthill-McKee to technika stosowana do permutacji macierzy rzadkich w celu zmniejszenia ich szerokości pasma. Szerokość pasma macierzy to różnica między największym a najmniejszym indeksem kolumny dla każdego wiersza, z maksimum wziętym dla wszystkich wierszy. Algorytm ten jest szczególnie przydatny w kontekście rozwiązywania układów równań liniowych, gdzie może pomóc zmniejszyć złożoność obliczeniową. Dokładny opis kroków algorytmu Cuthill-McKee na podstawie pseudokodu z wykładu:

1. Wybierz wierzchołek startowy s. Można to zrobić na wiele sposobów, ale często wybiera się wierzchołek o najniższym stopniu.
2. Wykonaj przeszukiwanie wszerz (BFS) zaczynając od wierzchołka s, dodając wierzchołki do kolejki w kolejności rosnącego stopnia.
3. Wyciągnij wierzchołek v z kolejki i dodaj go do listy ordering.
4. Dla każdego nieodwiedzonego sąsiada u wierzchołka v, dodaj u do kolejki.
5. Powtarzaj kroki 3-4, dopóki kolejka nie jest pusta.
6. Wynikowym porządkiem jest lista ordering.

3.1.3 Algorytm permutacji macierzy Reversed Cuthill-McKee

Odwrócony algorytm Cuthill-McKee to wariant algorytmu Cuthill-McKee, który polega na odwróceniu wygenerowanego porządku. Algorytm RCM ma na celu minimalizację "fill-in" podczas faktoryzacji LU macierzy.

- Algorytm CM zaczyna od dowolnego węzła i przeprowadza przeszukiwanie wszerz w celu utworzenia struktury poziomów, podczas gdy algorytm RCM zaczyna od ostatniego poziomu i działa wstecz.
- CM skupia się na redukcji bandwithu, podczas gdy RCM ma na celu zminimalizowanie wypełnienia podczas faktoryzacji.
- Wybór między CM a RCM może zależeć od konkretnego algorytmu numerycznego i charakterystyki rozwiązywanego problemu.

Wersja odwrócona moża charakteryzować się lepszym działaniem w przypadku niektórych macierzy w zależności od ich specyfikacji.

3.2 Pseudokody algorytmów

3.2.1 Pseudokod: Proces fill-in w macierzach rzadkich - $Fill - in$

Pseudokod dodawania inezerowych wyrazów macierzy, które powstają na wskutek odejmowania wierszy w procesie faktoryzacji.

Algorithm 2 Proces fill-in

```
procedure FillInProcess(A)                          ▷ A to macierz rzadka
    n ← rozmiar(A)                      ▷ Liczba wierszy i kolumn macierzy A
    G ← graf eliminacyjny dla A    ▷ Tworzymy graf eliminacyjny na podstawie A
    for i ← 1 to n do
        for j ← i + 1 to n do
            if A[i][j] ≠ 0 then        ▷ Jeśli element macierzy jest niezerowy
                G ← dodaj krawędź (i, j) do G   ▷ Tworzymy krawędź w grafie eliminacyjnym
                for k ← 1 to n do
                    if A[k][i] ≠ 0 and A[k][j] = 0 then    ▷ Fill-in w kolumnie j
                        A[k][j] ← A[k][i]
                    end if
                end for
            end if
        end for
    end for
end procedure
```

3.2.2 Pseudokod: Proces eliminacji na grafie nieskierownym - *Gaussian Elimination Undirected*

Pseudokod algorytmu eliminacji Gaussa na grafie nieskierownym.

Algorithm 3 Proces eliminacji na grafie nieskierownym

```
procedure ELIMINATIONPROCESSUNDIRECTEDGRAPH(V, E)    ▷ G(A) = {V, E}
    for k ← 1 to n − 1 do
        adj(k) ← {l : {k, l} ∈ E}    ▷ Zbiór sąsiedztwa wierzchołka k
        V ← V \ {k}    ▷ Usuń wierzchołek z eliminowanego wiersza
        E ← E \ {{k, l} : l ∈ adj(k)}    ▷ Usuń jego krawędzie
        E ← E ∪ {{x, y} : x ∈ adj(k), y ∈ adj(k)}    ▷ Połącz wierzchołki, które z nim sąsiadowały
    end for
end procedure
```

3.2.3 Pseudokod: Proces eliminacji na grafie skierowanym - *Gaussian Elimination Directed*

Pseudokod algorytmu eliminacji Gaussa na grafie skierownym.

Algorithm 4 Proces eliminacji na grafie skierowanym

```
procedure ELIMINATIONPROCESSDIRECTEDGRAPH(V, E)    ▷ G(A) = {V, E}
    for k ← 1 to n − 1 do
        succ(k) ← {l : (k, l) ∈ E}    ▷ Zbiór następców wierzchołka k
        pred(k) ← {l : (l, k) ∈ E}    ▷ Zbiór poprzedników wierzchołka k
        V ← V \ {k}    ▷ Usuń wierzchołek k
        E ← E \ {(k, l) : l ∈ succ(k)}    ▷ Usuń krawędzie wychodzące z k
        E ← E \ {(l, k) : l ∈ pred(k)}    ▷ Usuń krawędzie wchodzące do k
        E ← E ∪ {(x, y) : x ∈ pred(k), y ∈ succ(k)}    ▷ Dodaj krawędzie pomiędzy poprzednikami a następcami
    end for
end procedure
```

3.2.4 Pseudokod: permutacja macierzy macierzy - $Cuthll - McKee$

Pseudokod algorytmu permutacji macierzy Cuthll-McKee.

Algorithm 5 Algorytm Cuthill–McKee

```
procedure CuthillMcKee(A, n)    ▷ A to macierz symetryczna, n to rozmiar
    Znajdź obwodowy wierzchołek x o najniższym stopniu
    R ← {x}    ▷ Zainicjuj zbiór wynikowy pierwszym wierzchołkiem
    for i ← 1 to n do
        A_i ← Zbiór sąsiedztwa R_i    ▷ R_i to i-ty składnik zbioru R
        A_i ← A_i \ R    ▷ Wyklucz wierzchołki już zawarte w R
        Posortuj A_i rosnąco według minimalnego poprzednika i stopnia wierzchołka
        Dodaj A_i do zbioru wynikowego R
    end for
    return R    ▷ Uporządkowany n-krotka wierzchołków
end procedure
```

3.2.5 Pseudokod: permutacja macierzy macierzy - $Reversed\ Cuthll - McKee$

Pseudokod odwróconego algorytmu permutacji macierzy Reversed Cuthll-McKee.

Algorithm 6 Odwrócony algorytm Cuthill–McKee (Reversed Cuthill-McKee)

```
procedure ReversedCuthillMcKee(A, n)    ▷ A to macierz symetryczna, n to rozmiar
    Znajdź obwodowy wierzchołek x o najniższym stopniu
    R ← {x}    ▷ Zainicjuj zbiór wynikowy pierwszym wierzchołkiem
    for i ← 1 to n do
        A_i ← Zbiór sąsiedztwa R_i    ▷ R_i to i-ty składnik zbioru R
        A_i ← A_i \ R    ▷ Wyklucz wierzchołki już zawarte w R
        Posortuj A_i malejąco według maksymalnego poprzednika i stopnia wierzchołka
        Dodaj A_i do zbioru wynikowego R
    end for
    return R    ▷ Uporządkowany n-krotka wierzchołków
end procedure
```

3.2.6 Pseudokod: Algorytm permutacji macierzy z użyciem miary minimalnego stopnia wierzchołków - *Minimum Degree Permutation*

Pseudokod permutacji macierzy algorytmem Minimum Degree Permutation.

Algorithm 7 Minimum Degree Algorithm

```
procedure minimumDegreePermutation(G = (V, E))          ▷ Graf eliminacji
    while V is not empty do          ▷ Dopóki istnieją nieodwiedzone węzły
        p ← wybierz węzeł p z V o minimalnej liczbie sąsiadów ▷ "pivot"
        UpdateEliminationGraph(G, p)  ▷ Zaktualizuj graf eliminacji
        for all v ∈ sąsiedzi(p) do    ▷ Usuń wierzchołek p i zaktualizuj stopnie sąsiadów
            degree[v] ← degree[v] − 1
        end for
        Usuń p z V                    ▷ Oznacz wierzchołek jako odwiedzony
    end while
end procedure
procedure UpdateEliminationGraph(G, p)
    Usuń węzeł p z grafu eliminacji: V ← V \ {p}
    for all (u, v) ∈ E do  ▷ Usuń krawędzie związane z wierzchołkiem p
        if u = p or v = p then
            E ← E \ {(u, v)}
        end if
    end for
    for all u, v ∈ E do      ▷ Dodaj nowe krawędzie między sąsiadami p
        if u i v są sąsiadami p then
            E ← E ∪ {(u, v)}
        end if
    end for
end procedure
```

3.2.7 Pseudokod: Algorytm minimalnego stopnia wierzchołków na grafie eliminacyjnym - *Minimum Degree Elimination*

Pseudokod eliminacji minimalnego stopnia wierzchołków w grafie eliminacji algorytmem Minimum Degree Elimination.

Algorithm 8 Minimum Degree on Elimination Graph

```
procedure MINIMUMDEGREEELIMINATIONGRAPH(G_0 = (V, E))    ▷ Graf eliminacyjny
    G_k ← G_0                                            ▷ Inicjalizacja grafu eliminacyjnego
    for all v_i ∈ V do
        t_i ← liczba sąsiadów v_i w G_0
    end for
    for k ← 1 to n do
        p ← min_{t_i ∈ V_{G_{k-1}}} t_i                  ▷ Znajdź wierzchołek o minimalnym stopniu
        for all v_i ∈ adj_{G_{k-1}}(p) do                ▷ Dla każdego sąsiada v_i w G_{k-1}(p)
            adj_{G_{k-1}}(v_i) ← (adj_{G_{k-1}}(v_i) ∪ adj_{G_{k-1}}(p)) \ {v_i, p}    ▷ Aktualizuj zbiór sąsiadów
            t_i ← liczba sąsiadów v_i w G_k
        end for
        V_{G_k} ← V_{G_k} \ {p}                          ▷ Usuń wierzchołek p z V_{G_k}
    end for
end procedure
```

3.2.8 Pseudokod: Algorytm Cuthill-McKee z BFS - $Cuthill-McKeeBFS$

Pseudokod algorytmu Cuthill-McKee z BFS - wersja algorytmu z wykładu.

Algorithm 9 Algorytm Cuthill-McKee

```
procedure CuthillMcKeeBFS(G = (V, E))    ▷ Graf nieskierowany
    n ← liczba wierzchołków w G
    visited ← lista oznaczająca, czy wierzchołek był odwiedzony
    ordering ← lista przechowująca order wierzchołków w kolejności Cuthill-McKee
    degree ← lista stopni wierzchołków
    for all v ∈ V do
        visited[v] ← false
        degree[v] ← stopień wierzchołka v w G
    end for
    for all v ∈ V do
        if not visited[v] then
            BFS(v, visited, ordering, degree)
        end if
    end for
    return ordering
end procedure
procedure BFS(v, visited, ordering, degree)
    queue ← pusta kolejka
    visited[v] ← true
    enqueue(queue, v)
    while not isEmpty(queue) do
        u ← dequeue(queue)
        append(ordering, u)
        for all w ∈ sąsiedzi(u) posortowani rosnąco według degree do
            if not visited[w] then
                visited[w] ← true
                enqueue(queue, w)
            end if
        end for
    end while
end procedure
```

3.3 Fragmenty kodu programu realizującego zadanie

3.3.1 Importy

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse.linalg import svds
from matplotlib.colors import ListedColormap
from math import inf
from queue import Queue
```

3.3.2 Reprezentacja macierzy 3D

```
def create_3d_matrix(k):
    n = 2 **(3*k)
    matrix = np.zeros((n, n))
    for i in range(n):
        for j in [0, 1, -1, 2**(k), -2**(k), 2**(k*2), -2**(k*2)]:
            if i + j >= 0 and i + j < n:
                matrix[i][i + j] = np.random.random()
    return matrix
```

3.3.3 Struktura drzewa

```
class Node:
    def __init__(self):
        self.rows = None
        self.columns = None
        self.matrix = None
        self.rank = None
        self.U = None
        self.D = None
        self.VT = None
        self.children = []
```

3.3.4 Kompresja

```
def split_matrix(X):
    n = X.shape[1]//2
    return X[:n, :n], X[:n, n:], X[n:, :n], X[n:, n:]

def compress(M, r, epsilon):
    if not np.any(M):
        node = Node()
        node.rank = 0
        node.rows, node.columns = M.shape
        return node

    if M.shape[0] <= r + 1 and M.shape[1] <= r + 1:
        node = Node()
        node.rows, node.columns = M.shape
        node.matrix = M
        return node
```

```
    U, D, VT = svds(M, k= r + 1)

    if abs(D[0]) < epsilon:
        node = Node()
        node.rows, node.columns = M.shape
        node.rank = r
        node.U = U[:, 1:]
        node.D = D[1:]
        node.VT = VT[1:]

    else:
        M_11, M_12, M_21, M_22 = split_matrix(M)
        node = Node()
        node.rows, node.columns = M.shape
        node.rank = None
        node.children.append(compress(M_11, r, epsilon))
        node.children.append(compress(M_12, r, epsilon))
        node.children.append(compress(M_21, r, epsilon))
        node.children.append(compress(M_22, r, epsilon))

    return node
```

3.3.5 Wizualizacja

```
# Macierzy 2D:
def draw_matrix(M, title):
    fig, ax = plt.subplots()
    ax.set_title(title)
    ax.spy(M)
    plt.show()

# Skompresowanej macierzy przechowywanej w strukturze drzewa
def create_plot(node):
    if node.rank is not None:
        matrix = np.zeros((node.rows, node.columns))
        if node.rank > 0:
            matrix[:, :node.rank] = 1
            matrix[:node.rank, :] = 1

        return matrix

    elif node.matrix is not None:
        return np.ones((node.rows, node.columns))

    else:
        return np.vstack((
                np.hstack((create_plot(node.children[0]),
                           create_plot(node.children[1]))),
                np.hstack((create_plot(node.children[2]),
                           create_plot(node.children[3]))),))

def draw_node_matrix(node, title):
    fig, ax = plt.subplots()
    ax.set_title(title)
    ax.matshow(create_plot(node), cmap=ListedColormap(['w', 'brown']))
    plt.show()
```

3.3.6 Algorytm Minimum Degree

```
def minimum_degree_permutation(matrix):
    result_matrix = matrix.copy()

    rows, columns = matrix.shape
    adj_matrix = {i:set() for i in range(rows)}

    for i in range(rows):
        for j in range(columns):
            if matrix[i][j] != 0 and i != j:
                adj_matrix[i].add(j)

    permutation = []
    for i in range(rows):
        deg_min = columns + 1
        for v, adj in adj_matrix.items():
            if len(adj) < deg_min:
                v_min = v
                deg_min = len(adj)
        for v in adj_matrix:
            adj_matrix[v] = adj_matrix[v].difference([v_min])
        for u in adj_matrix[v_min]:
            adj_matrix[u] = (adj_matrix[u].union(adj_matrix[v_min].difference([u])))
        adj_matrix.pop(v_min)
        permutation.append(v_min)

    for i in range(len(permutation)):
        if i != permutation[i]:
            result_matrix[i,:] = matrix[permutation[i],:].copy()
    matrix = result_matrix.copy()
    for i in range(len(permutation)):
         if i != permutation[i]:
            result_matrix[:,i] = matrix[:,permutation[i]].copy()

    return result_matrix
```

3.3.7 Algorytm Cuthill-McKee

```
def cuthill_mckee_permutation(matrix):
    result_matrix = matrix.copy()
    n, m = matrix.shape

    adj_matrix = {i:set() for i in range(n)}

    for i in range(n):
        for j in range(m):
            if matrix[i][j] != 0 and i != j:
                adj_matrix[i].add(j)
    deg_min = inf

    for v, adj in adj_matrix.items():
        if len(adj) < deg_min:
            v_min = v
            deg_min = len(adj)
    permuattion = []
    pred = [inf for _ in range(n)]
    visited = [False for _ in range(n)]
    pred[v_min] = 0
```

```
    queue = Queue()
    queue.put((0, len(adj_matrix[v_min]), v_min))

    while not queue.empty():
        _, _, v = queue.get()
        if not visited[v]:
            visited[v] = True
            p = len(permuattion)
            permuattion.append(v)
            for u in adj_matrix[v]:
                if visited[u] == False:
                    if pred[u] > p:
                        pred[u] = p
                    queue.put((pred[u], len(adj_matrix[u]), u))

    for i in range(len(permuattion)):
        if i != permuattion[i]:
            result_matrix[i,:] = matrix[permuattion[i],:].copy()
    matrix = result_matrix.copy()
    for i in range(len(permuattion)):
         if i != permuattion[i]:
            result_matrix[:,i] = matrix[:,permuattion[i]].copy()

    return result_matrix
```

3.3.8 Algorytm Reversed Cuthill-McKee

```
def reversed_cuthill_mckee_permuatation(matrix):
    return cuthill_mckee_permutation(matrix)[::-1]


for k in [2, 3, 4]:
    matrix = create_3d_matrix(k)

    draw_matrix(matrix, f"k={k} - wzorzec rzadkosci")
    draw_node_matrix(compress(matrix, 1, 0.0000001),
        f"k={k} - macierz rzadka po kompresji")

    matrix_after_permutation = reversed_cuthill_mckee_permuatation(matrix)

    draw_matrix(matrix_after_permutation,
        f"k={k} - wzorzec rzadkosci po permutacji Reversed Cuthill-McKee")
    draw_node_matrix(compress(matrix_after_permutation, 1, 0.0000001),
        f"k={k} - macierz po permutacji Reversed Cuthill-McKee i kompresji")
```

4 Wzorce rzadkości (sparsity patern) macierzy rzadkich - oryginalne oraz po operacjach kompresji i permutacji

Przygotowano wzorzec rzadkości (sparsity patern) macierzy rzadkiej przed kompresją i permutacją. W tym celu posłużono się środowiskiem MATLAB i wykorzystano:

$$spy(A),$$

gdzie A to macierz (w naszym wypadku rzadka).

4.1 MATLAB

MATLAB (MATrix LABoratory) to zaawansowane środowisko programistyczne oraz język programowania używany głównie do manipulacji i analizy danych numerycznych, zwłaszcza macierzowych. MATLAB oferuje bogatą gamę narzędzi do przetwarzania sygnałów, grafiki, algorytmów numerycznych, analizy danych, uczenia maszynowego i innych dziedzin.

Rysunek 2: Logo MATLABa

4.1.1 Wykorzystanie MATLABa w laboratorium

Przygotowane funkcje MATLAB do generowania wzorców rzadkości, kompresji i permutacji.

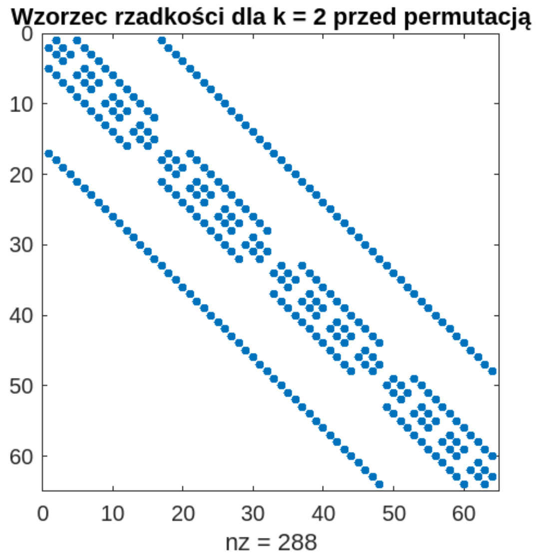
```
for k = 2:4
    dim = 2^k; % Wymiary siatki
    n = dim^3; % Liczba wierzchoĆk w
    A = sparse(n, n); % Pusta macierz rzadka

    for x = 1:dim
        for y = 1:dim
            for z = 1:dim
                idx = (x-1)*dim^2 + (y-1)*dim + z;
                % Dodawanie krawŹdzi do s siednich
                    wierzchoĆk w
                if x > 1, A(idx, idx-dim^2) = rand;
                   end
                if x < dim, A(idx, idx+dim^2) = rand
                   ; end
                if y > 1, A(idx, idx-dim) = rand;
                   end
                if y < dim, A(idx, idx+dim) = rand;
                   end
                if z > 1, A(idx, idx-1) = rand; end
                if z < dim, A(idx, idx+1) = rand;
                   end
            end
        end
    end
```

```
    % Przed permutacj
    figure;
    spy(A);
    title(['Wzorzec rzadkoŻci dla k = ', num2str(k)
       , ' przed permutacj ']);

    % Cuthill-McKee
    p_cm = symrcm(A);
    A_cm = A(p_cm,p_cm);
    figure;
    spy(A_cm);
    title(['Wzorzec rzadkoŻci dla k = ', num2str(k)
       , ' | Cuthill-McKee']);

    % Minimum Degree
    p_md = colamd(A);
    A_md = A(p_md,p_md);
    figure;
    spy(A_md);
    title(['Wzorzec rzadkoŻci dla k = ', num2str(k)
       , ' | Minimum Degree']);
end
```

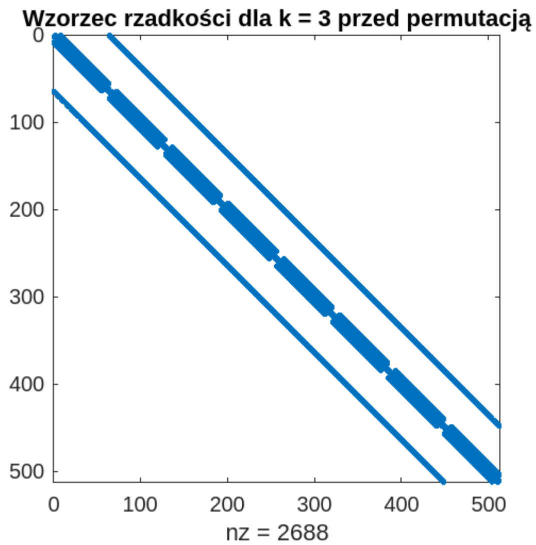
4.2 Wzorce rzadkości w MATLABie

Przygotowano wzorce rzadkości za pomocą narzędzi w środowiku MATLAB.

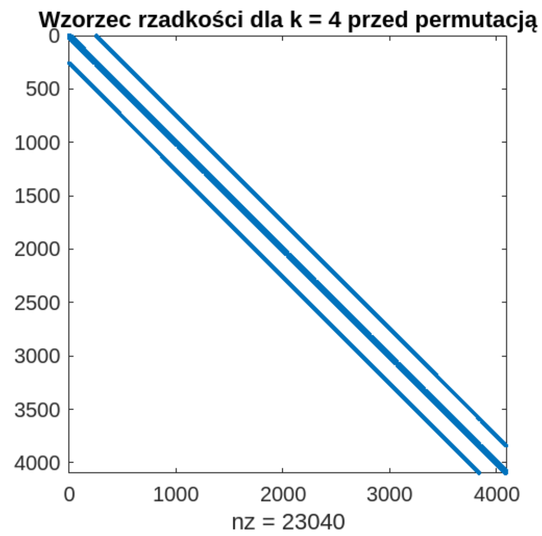
4.2.1 Oryginalny wzorzec rzadkości przed kompresją i permutacją



Rysunek 3: Oryginalny wzorzec przed kompresją i permutacją dla $k = 2$

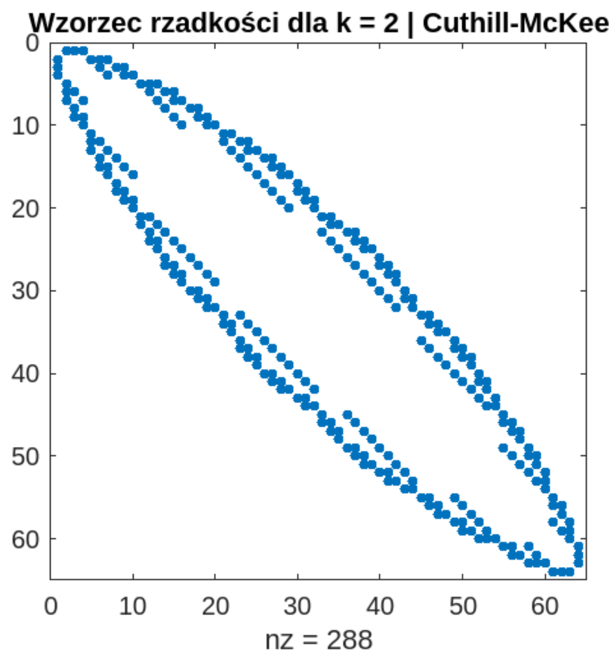


Rysunek 4: Oryginalny wzorzec przed kompresją i permutacją dla $k = 3$

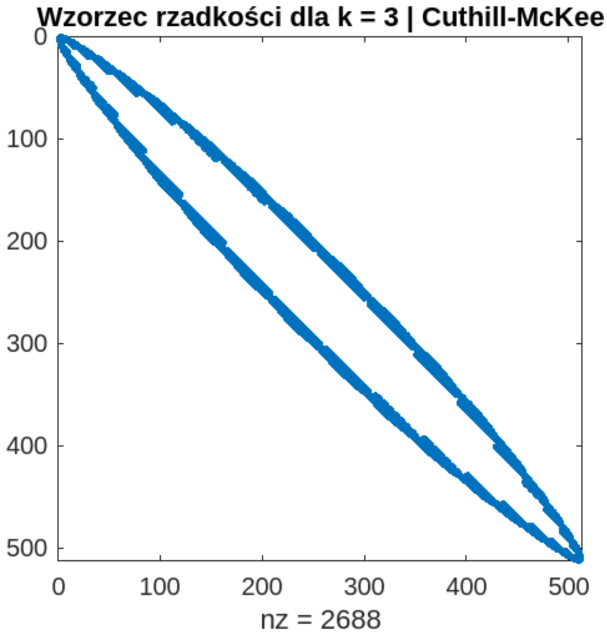


Rysunek 5: Oryginalny wzorzec przed kompresją i permutacją dla $k = 4$

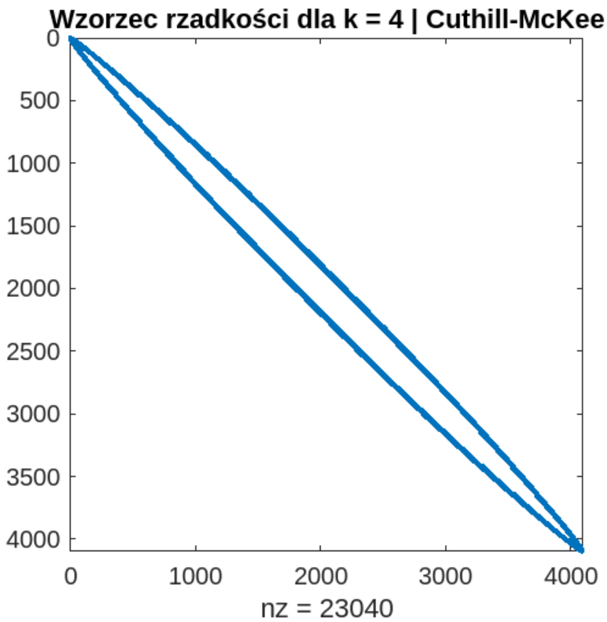
4.2.2 Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Cuthill-McKee



Rysunek 6: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Cuthill-McKee dla $k = 2$

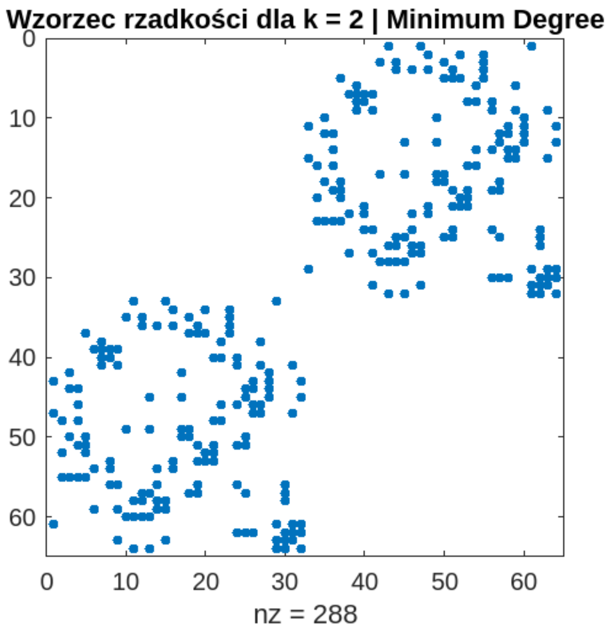


Rysunek 7: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Cuthill-McKee dla $k = 3$

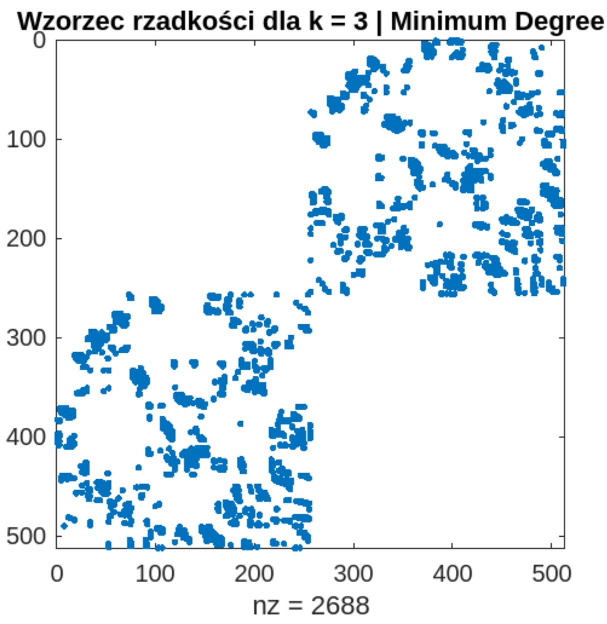


Rysunek 8: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Cuthill-McKee dla $k = 4$

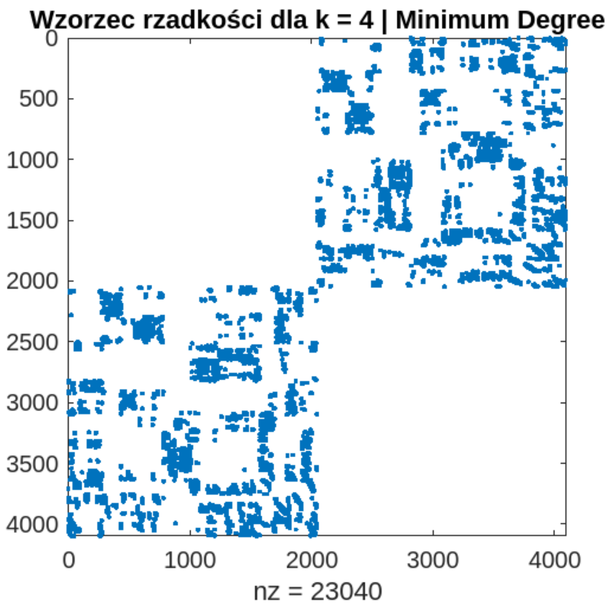
4.2.3 Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Minimum Degree



Rysunek 9: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Minimum dla $k = 2$



Rysunek 10: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Minimum dla $k = 3$

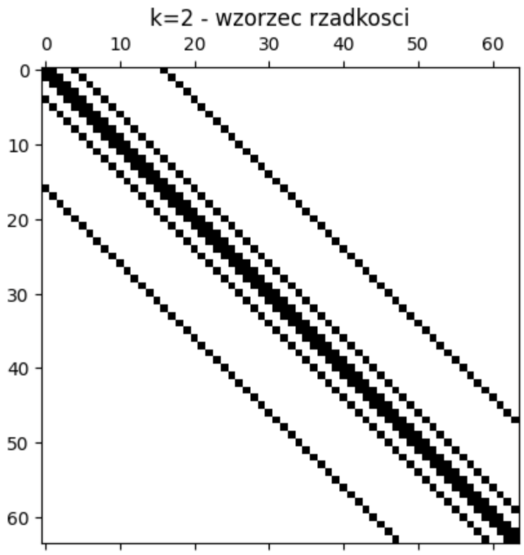


Rysunek 11: Oryginalny wzorzec rzadkości przed kompresją i po permutacji algorytmem Minimum dla $k = 4$

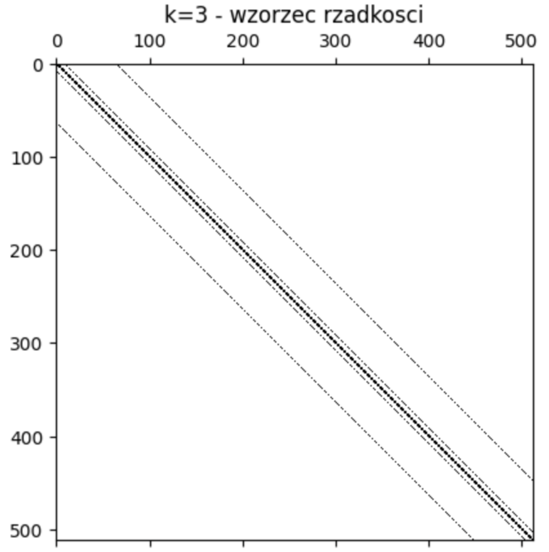
4.3 Wzorce rzadkości - rysowacz macierzy hierarchicznych

Przygotowano wzorce rzadkości za pomocą zaimplementowanego rysowacza macierzy hierarchicznych.

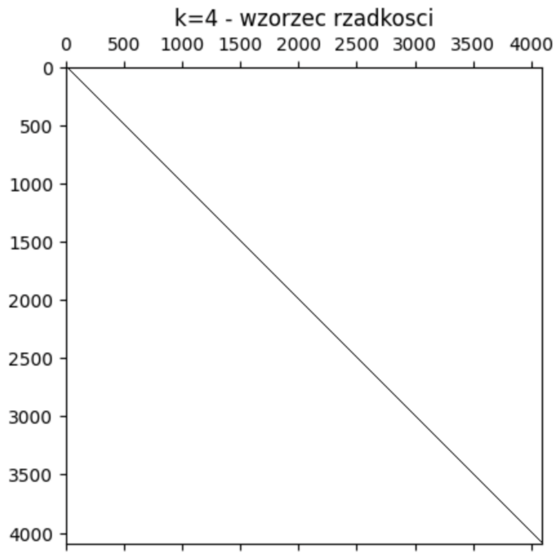
4.3.1 Wzorzec rzadkości przed kompresją i permutacją



Rysunek 12: Wzorzec rzadkości przed kompresją i permutacją dla $k = 2$

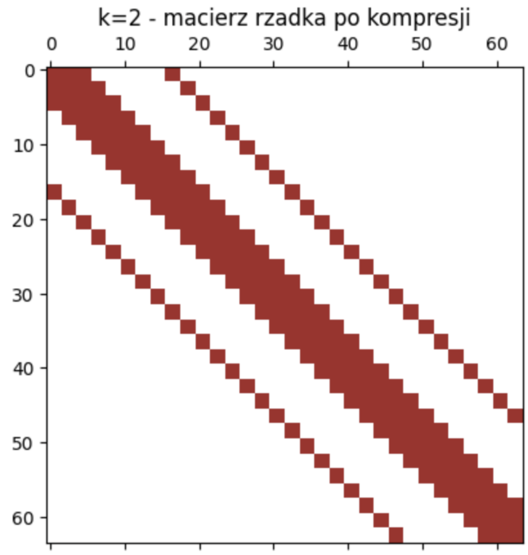


Rysunek 13: Wzorzec rzadkości przed kompresją i permutacją dla $k = 3$

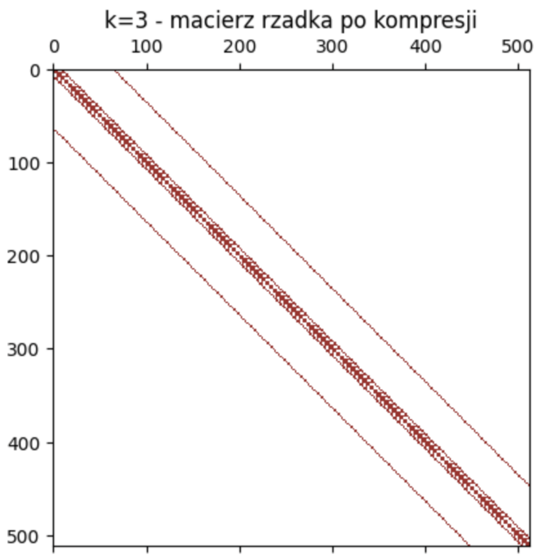


Rysunek 14: Wzorzec rzadkości przed kompresją i permutacją dla $k = 4$

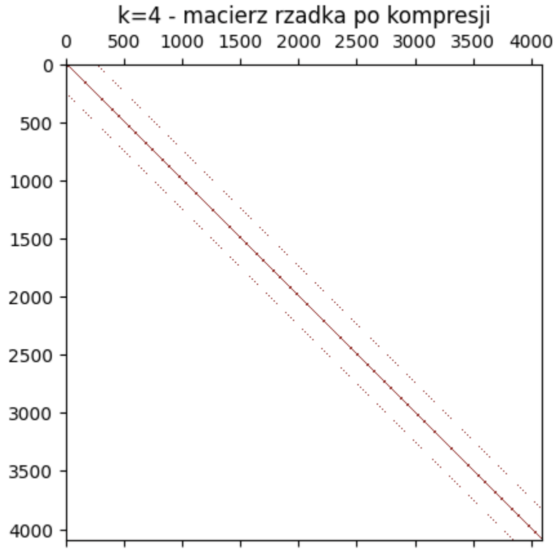
4.3.2 Wzorzec rzadkości po kompresji i przed permutacją



Rysunek 15: Wzorzec rzadkości po kompresji i przed permutacją dla $k = 2$

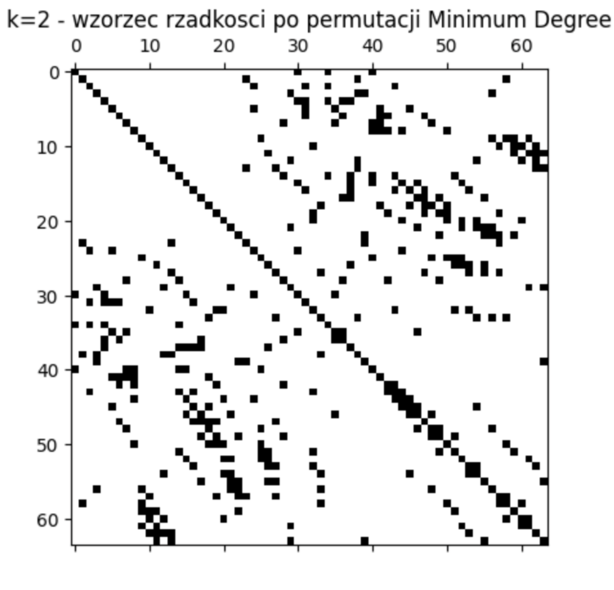


Rysunek 16: Wzorzec rzadkości po kompresji i przed permutacją dla $k = 3$

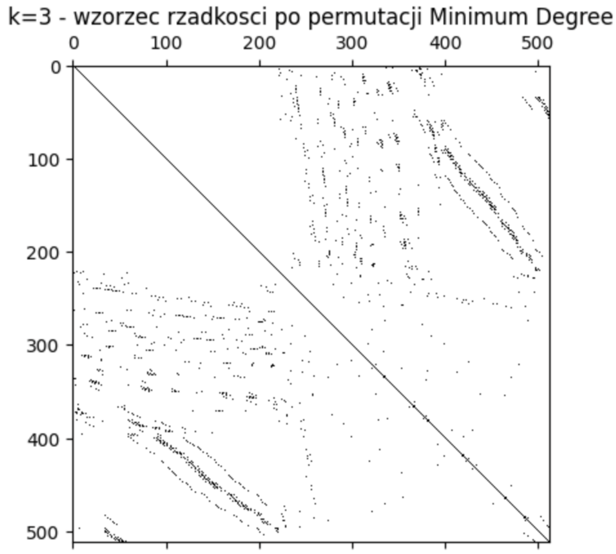


Rysunek 17: Wzorzec rzadkości po kompresji i przed permutacją dla $k = 4$

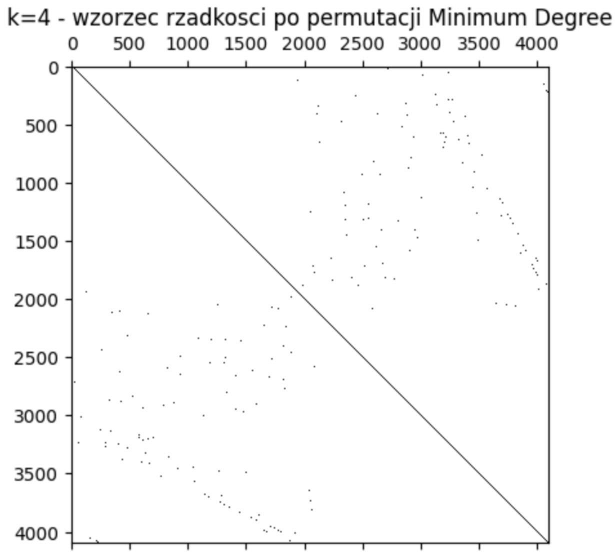
4.3.3 Wzorzec rzadkości przed kompresją i po permutacji Minimum Degree



Rysunek 18: Wzorzec rzadkości przed kompresją i po permutacji Minimum Degree dla $k = 2$

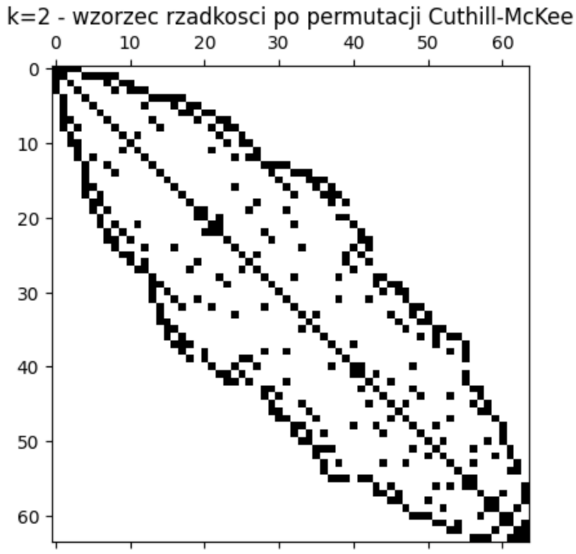


Rysunek 19: Wzorzec rzadkości przed kompresją i po permutacji Minimum Degree dla $k = 3$

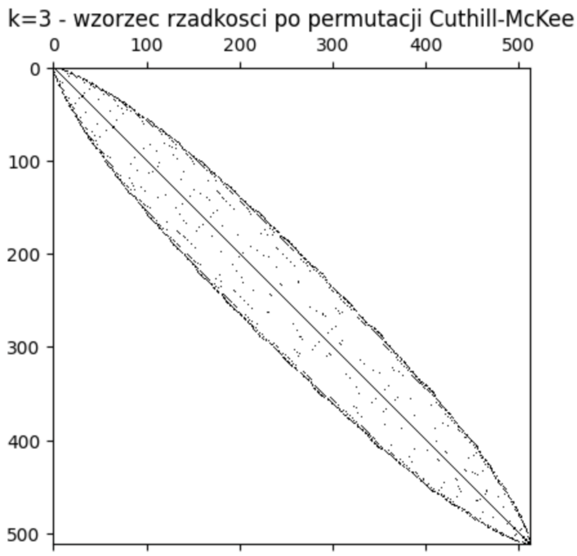


Rysunek 20: Wzorzec rzadkości przed kompresją i po permutacji Minimum Degree dla $k = 4$

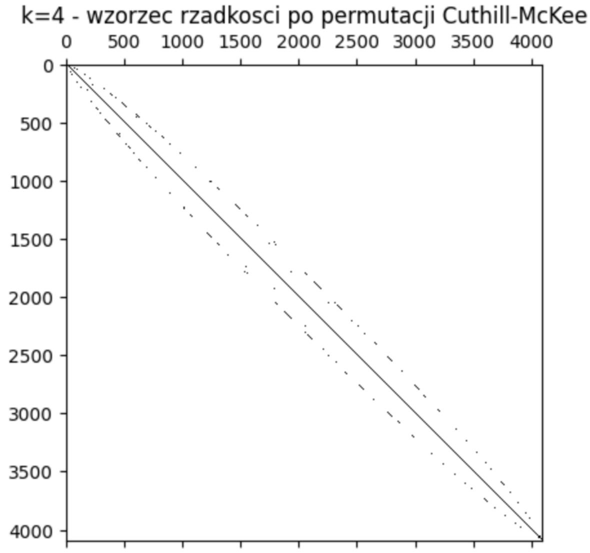
4.3.4 Wzorzec rzadkości przed kompresją i po permutacji Cuthill-McKee



Rysunek 21: Wzorzec rzadkości przed kompresją i po permutacji Cuthill-McKee dla $k = 2$

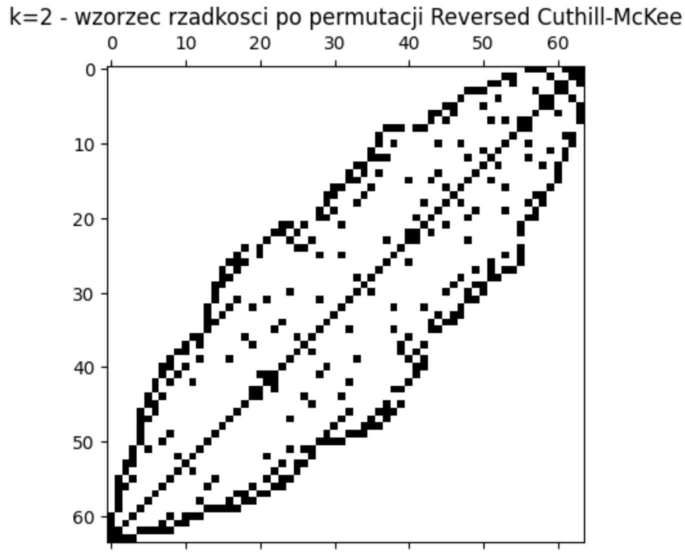


Rysunek 22: Wzorzec rzadkości przed kompresją i po permutacji Cuthill-McKee dla $k = 3$

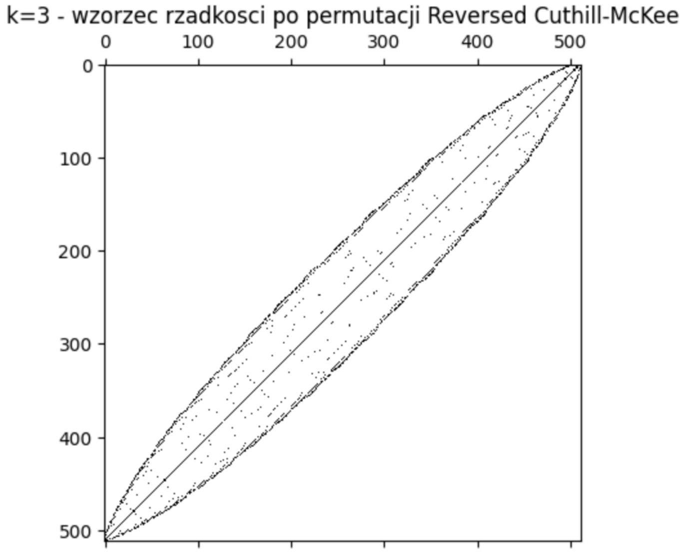


Rysunek 23: Wzorzec rzadkości przed kompresją i po permutacji Cuthill-McKee dla $k = 4$

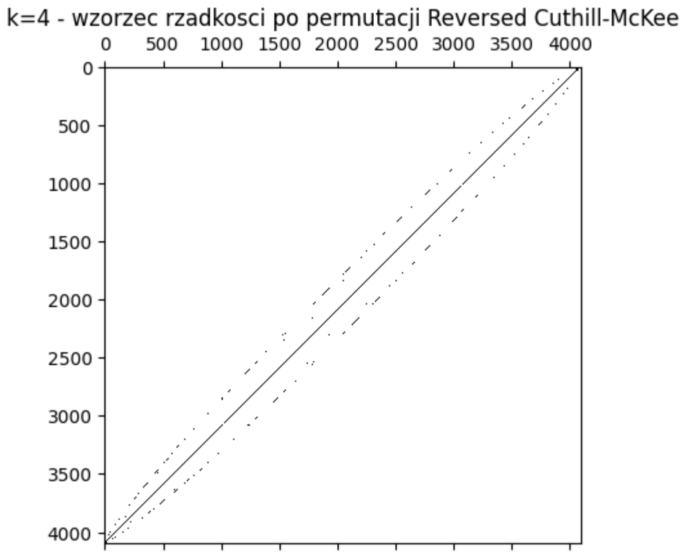
4.3.5 Wzorzec rzadkości przed kompresją i po permutacji Reversed Cuthill-McKee



Rysunek 24: Wzorzec rzadkości przed kompresją i po permutacji Reversed Cuthill-McKee dla $k = 2$

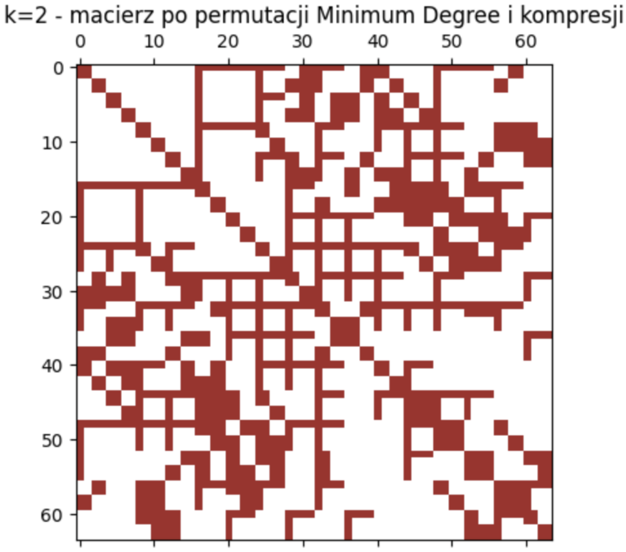


Rysunek 25: Wzorzec rzadkości przed kompresją i po permutacji Reversed Cuthill-McKee dla $k = 3$

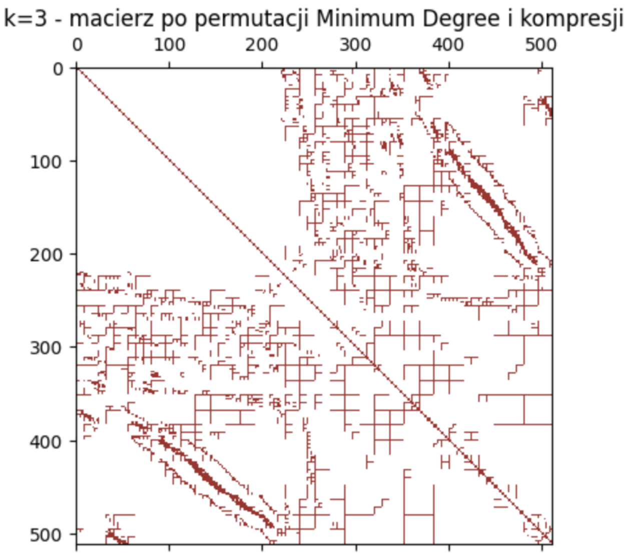


Rysunek 26: Wzorzec rzadkości przed kompresją i po permutacji Reversed Cuthill-McKee dla $k = 4$

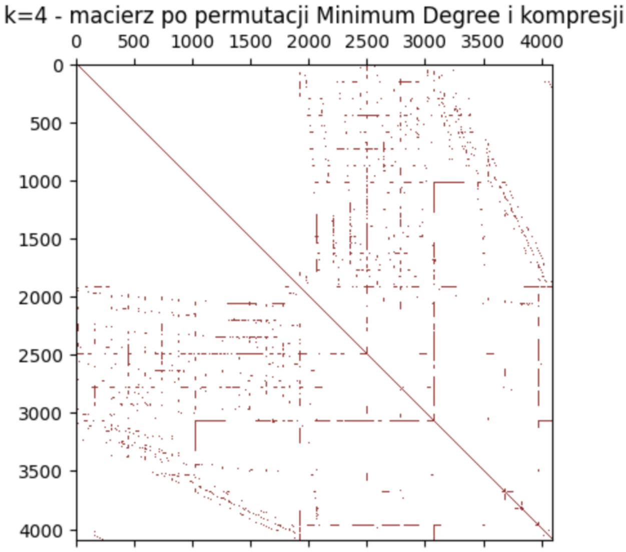
4.3.6 Wzorzec rzadkości po kompresji i permutacji Minimum Degree



Rysunek 27: Wzorzec rzadkości po kompresji i permutacji Minimum Degree dla $k = 2$

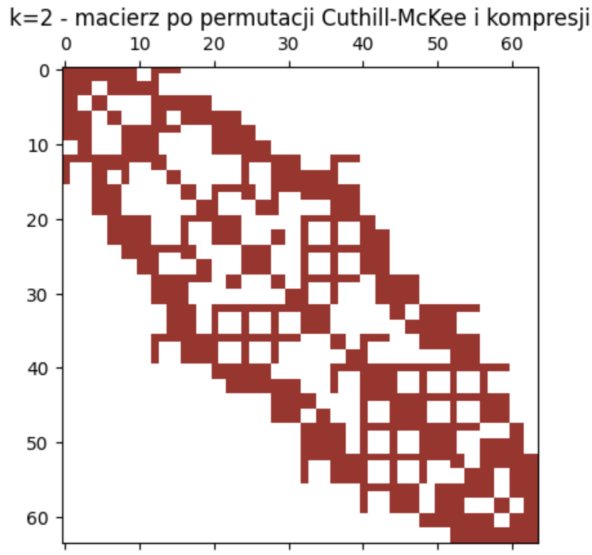


Rysunek 28: Wzorzec rzadkości po kompresji i permutacji Minimum Degree dla $k = 3$

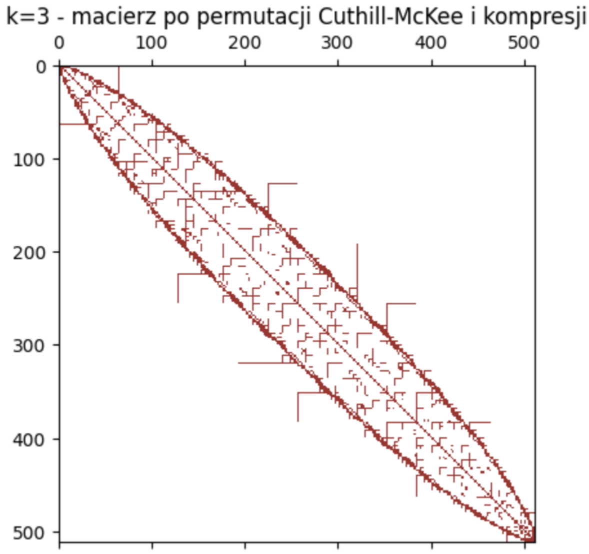


Rysunek 29: Wzorzec rzadkości po kompresji i permutacji Minimum Degree dla $k = 4$

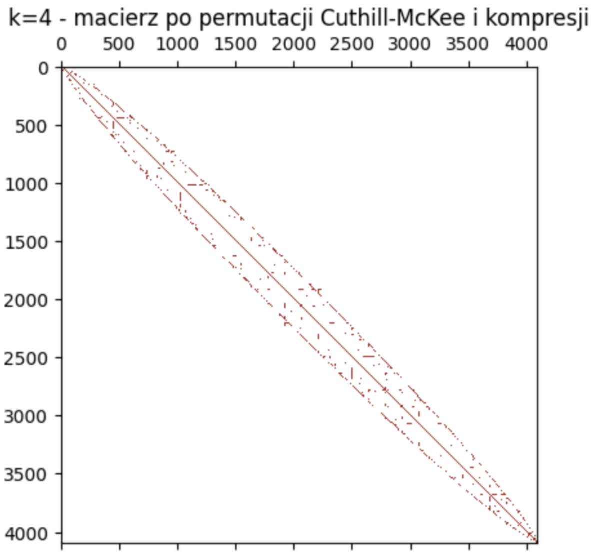
4.3.7 Wzorzec rzadkości po kompresji i permutacji Cuthill-McKee



Rysunek 30: Wzorzec rzadkości po kompresji i permutacji Cuthill-McKee dla $k = 2$

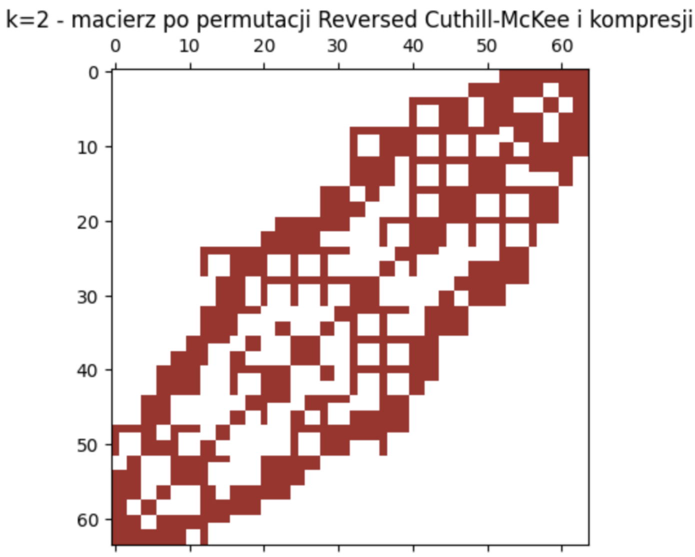


Rysunek 31: Wzorzec rzadkości po kompresji i permutacji Cuthill-McKee dla $k = 3$

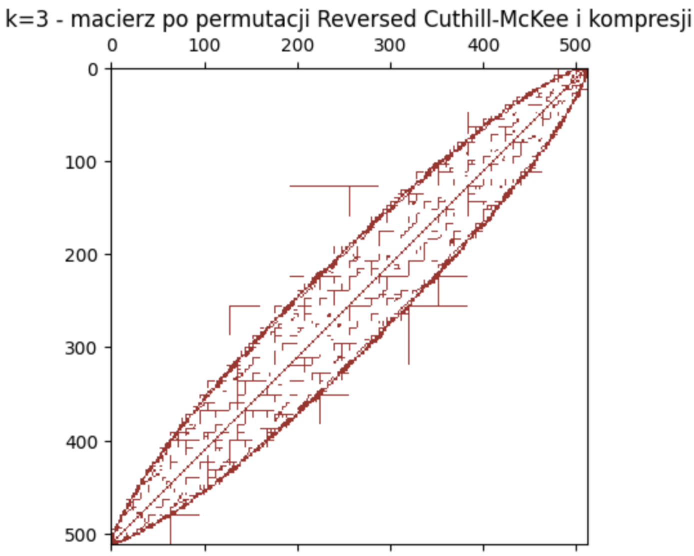


Rysunek 32: Wzorzec rzadkości po kompresji i permutacji Cuthill-McKee dla $k = 4$

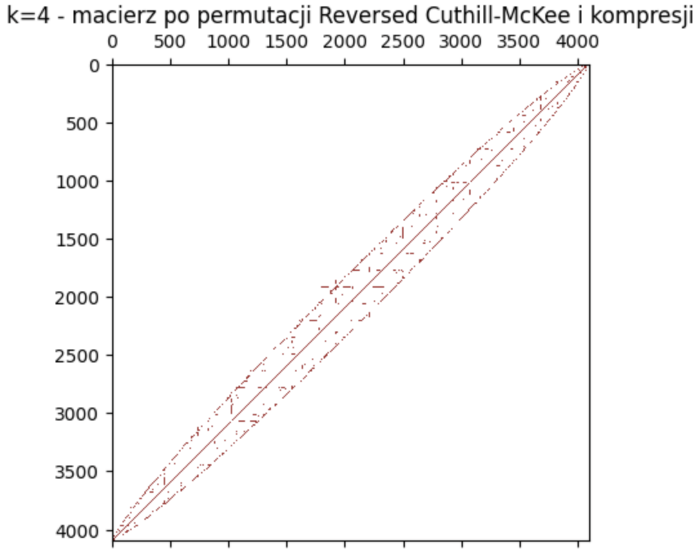
4.3.8 Wzorzec rzadkości po kompresji i permutacji Reversed Cuthill-McKee



Rysunek 33: Wzorzec rzadkości po kompresji i permutacji Reversed Cuthill-McKee dla $k = 2$



Rysunek 34: Wzorzec rzadkości po kompresji i permutacji Reversed Cuthill-McKee dla $k = 3$



Rysunek 35: Wzorzec rzadkości po kompresji i permutacji Reversed Cuthill-McKee dla $k = 4$

5 Wnioski

LABORATORIUM pozwoliło dobrze zrozumieć zagadnienie permutacji macierzy oraz kompresji macierzy rzadkich przed i po permutacji, a także dało dobry ogląd na strukturalne własności macierzy i możliwości ich reprezentacji oraz działania na różnych ich reprezentacjach.

Laboratorium pozwoliło na zrozumienie roli algorytmów macierzowych, ich implementacji w MATLABie oraz korzyści płynących z permutacji i kompresji macierzy w kontekście analizy numerycznej. Implementacja tych technik może znacznie poprawić efektywność obliczeń numerycznych przy jednoczesnym optymalnym wykorzystaniu zasobów pamięciowych i obliczeniowych.

5.1 Refleksje płynące z laboratirum

- Wykorzystanie MATLABa w analizie numerycznej
  - MatLab okazuje się potężnym narzędziem do implementacji i testowania algorytmów numerycznych na macierzach.
  - Intuicyjna składnia i szeroki zakres funkcji Matlaba ułatwiają eksperymentowanie z różnymi algorytmami i analizowanie wyników.
- Kompresja macierzy
  - Kompresja macierzy jest istotnym elementem, zwłaszcza gdy macierze są rzadkie.
  - Odpowiednia implementacja kompresji pozwala na efektywne przechowywanie i przetwarzanie dużych macierzy, co jest istotne w przypadku problemów z dużą ilością danych.
- Wykorzystanie i właściwości algorytmów permutacji macierzy
  - Niektóre algorytmy mogą być bardziej lub mniej wrażliwe na specyfikę danych wejściowych. Warto zbadać, jak wyniki zmieniają się dla różnych zestawów danych.
  - Wybór konkretnego algorytmu powinien zależeć od konkretnych potrzeb aplikacji. W niektórych przypadkach warto zrezygnować z minimalizacji fill-in na rzecz szybkości wykonania.
  - Algorytmy różnią się pod względem złożoności obliczeniowej. Minimum Degree może być bardziej złożone obliczeniowo, ale jednocześnie generować bardziej optymalne wyniki.

- Efektywność algorytmów może zależeć od konkretnej struktury macierzy wejściowej. W niektórych przypadkach jeden z algorytmów może działać lepiej niż pozostałe.
- Minimum Degree okazał się skuteczny w minimalizowaniu fill-in, co może być istotne dla algorytmów rozwiązywania układów równań liniowych.
- Algorytmy Cuthill-McKee i Reversed Cuthill-McKee skutkowały znaczącą redukcją bandwidth macierzy, co może być kluczowe dla efektywnej pracy algorytmów numerycznych.
- AQlgorytm Minimum jest heurystyką, która nie musi dawać optymalnych rozwiązań, ale cześto są one zadowalające i pozwalają na redukcję złożoności obliczeniowej.

5.2 Podsumowanie

Laboratorium pozwoliło na dobre zrozumienie działania algorytmów permutacji macierzy oraz pozwoliło na zdobycie dodatkowych umiejętności z wykorzystania narzędzia MATLAB. Kompresja macierzy przed i po permutacji dała wgląd w działanie i włsności operacji i działań na macierzach. Z laboratorium i wykładu udało się wynieść bardzo dużo wiedzy i umiejętności oraz bliżej poznać zagadnienia związane z komputerowym działaniem i przetwarzaniem macierzy.

6 Refleksja

6.1 Wiersz o macierzach

W macierzach liczby snują taniec swój,
Symfonia liczb, jak w życiu dźwięk.
Istnieją reguły, wzory jak wiersz,
Gdzie mnożą się plany, jak słowa w mowie.

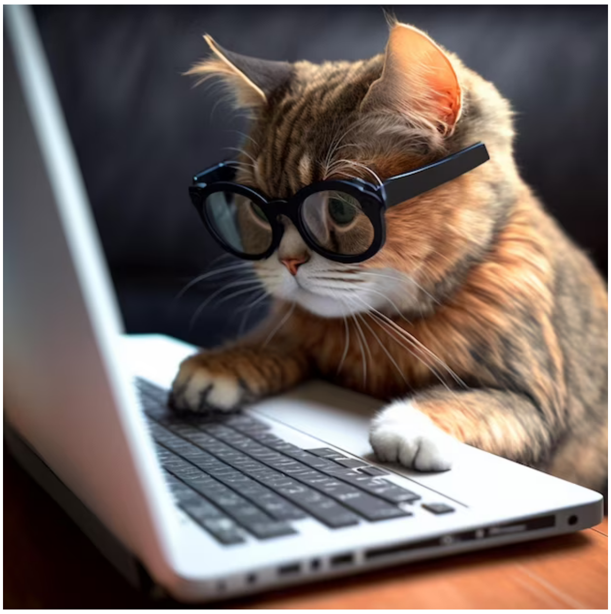
O szlachetne macierze, kunsztu tajemne,
W ich komórkach życie, jak wiersza rym.
Dodawajmy do nich, jak dni do życia,
By sumy były pełne, niczym harmonia.

Mnożenie jak pasja, rozmnożenie planów,
W macierzach kryje się siła, jak wzdłuż linii stan.
Kolumny i wiersze, jak porządki życia,
Niech równania snują, jak losy rozwijan.

I w obliczeniach skryte jest zdrowie,
Jak w macierzach równowaga, harmonia w sobie.
Dodajmy więc wartość, by sumy kwitły,
I życia macierz niech nam błogosławi.

W macierzach ukryte są tajemnice świata,
Jak zdrowie ukryte, jak skarb w skrzyni złota.
Niech te słowa brzmią, jak wiersz o zdrowiu szlachetnym,
By macierze życia nas prowadziły nieustannie.

6.2 Symboliczna reprezentacja



Rysunek 36: Symboliczna reprezentacja studentów odkrywających piękno matematyki, w szczególności macierzy i algorytmów macierzowych...

Literatura

[1] Wykłady i laboratoria prowadzone przez Pana prof. dr hab. Macieja Paszyńskiego

[2] Mathworks: Reversed Cuthill-McKee

[3] Wikipedia: Cuthill-McKee

[4] Wikipedia: Graph bandwith

[5] Wikipedia: Sparse matrix

***