Teoria Śladów dla Algorytmu Eliminacji Gaussa

Antoni Dulewicz

Spis treści

1 Wprowadzenie

Zadanie przedstawia analizę algorytmu eliminacji Gaussa w kontekście teorii śladów. Dokument opisuje lokalizację czynności niepodzielne, konstruowanie relacji zależności, reprezentację w postaci ciągu symboli oraz grafu Diekerta, a także przekształcenie do postaci normalnej Foaty. Poniżej znajduje się przykładowa macierz rozmiaru 3x3 dla której stworzymy graf Dickerta w wyprowadzeniu teoretycznym.

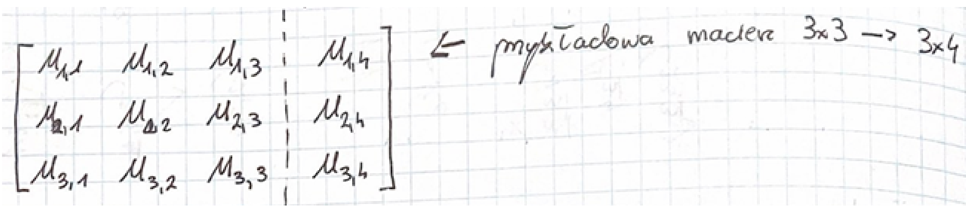

Figure 1: Enter Caption

2 Wyprowadzenie teotretyczne

2.1 Czynności niepodzielne oraz alfabet

- $A_{i,k}$: Znalezienie mnożnika dla wiersza i, aby odjąć go od wiersza k:

$$m_{k,i} = \frac{M_{k,i}}{M_{i,i}}$$

- $B_{i,j,k}$: Pomnożenie j-tego elementu wiersza i przez mnożnik, aby odjąć go od k-tego wiersza:

$$n_{k,i} = M_{i,j} \cdot m_{k,i}$$

- $C_{i,j,k}$: Odjęcie j-tego elementu wiersza i od wiersza k:

$$M_{k,j} = M_{k,j} - n_{k,i}$$

n - rozmiar macierzy.
Na naszym przykładzie ($n = 3$):

$$\Sigma = \{A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, \ldots, A_{1,3}, B_{1,1,3}, \ldots, B_{2,4,3}, C_{2,4,3}\}.$$

Ogólnie::

$$A = \{A_{i,k} \mid 1 \leq i \leq n,\ i < k \leq r\},$$
$$B = \{B_{i,j,k} \mid 1 \leq i \leq n,\ i \leq j \leq n+1,\ i \leq k \leq n\},$$
$$C = \{C_{i,j,k} \mid 1 \leq i < n,\ i \leq j \leq n+1,\ i < k \leq n\}.$$

$$\Sigma = A \cup B \cup C$$

2.2 Relacje zależności i niezależności

- Relacja zależności: $D = sym\{(D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5)^+\} \cup I_\Sigma$

$$D_1 = \{(A_{i,k}, B_{i,j,k}) \mid A_{i,k}, B_{i,j,k} \in \Sigma\},$$

$$D_2 = \{(B_{i,j,k}, C_{i,j,k}) \mid B_{i,j,k}, C_{i,j,k} \in \Sigma\},$$

$$D_3 = \{(C_{i_c,j_c,k_c}, A_{i_a,k_a}) \mid C_{i_c,j_c,k_c}, A_{i_a,k_a} \in \Sigma \wedge (j_c = i_a) \wedge [(k_c = i_a \vee i_b = k_c)]\},$$

(j_c = i_a) - są w tej samej kolumnie

(k_c = i_a) - wiersz, od którego odejmujemy w C,

jest taki sam jak wiersz dla którego znajdujemy mnożnik w A

(i_b = k_c) - wiersz od którego odejmujemy w C

jest taki sam jak wiersz od którego będziemy odejmować w A

$$D_4 = \{(B_{i_b,j,k_b}, C_{i_c,j,k_c}) \mid B_{i_b,j,k_b}, C_{i_c,j,k_c} \in \Sigma \wedge i_b = k_c\},$$

takie samo j - ta sama kolumna

(i_b = k_c) - wiersz który jest przemnażany w B

ten sam jak wiersz od którego odejmowaliśmy w C

$$D_5 = \{(C_{i_1,j,k}, C_{i_2,j,k}) \mid C_{i_1,j,k}, C_{i_2,j,k} \in \Sigma\}$$

- Relacja niezależności: $I = \Sigma - D$

2.3 Algorytm jako ciąg symboli

Wyzerowanie $M_{i,k}$:

$$A_{i,k}, B_{i,i,k}, C_{i,i,k}, B_{i,i+1,k}, \ldots, C_{i,n+1,k}$$

Algorytm eliminacji Gaussa polega na kilkukrotnym wyzerowaniu $M_{i,k}$, gdzie (i,k) należy do zbioru:

$$\{(1,2),(1,3),\ldots,(1,n+1),(2,3),(2,4),\ldots,(2,n+1),\ldots,(n,n+1)\}$$

Załóżmy, że operacja wyzerowania $M_{i,k}$ to ciąg operacji zapisany jako $s_{i,k}$. Wtedy algorytm eliminacji Gaussa A można zapisać jako:

$$A = (s_{1,2}, s_{1,3}, \ldots, s_{1,n+1}, s_{2,3}, s_{2,4}, \ldots, s_{2,n+1}, \ldots, s_{n,n+1})$$

2.4 Graf Diekerta

Wygenerowany graf zależności Diekerta wygląda następująco:

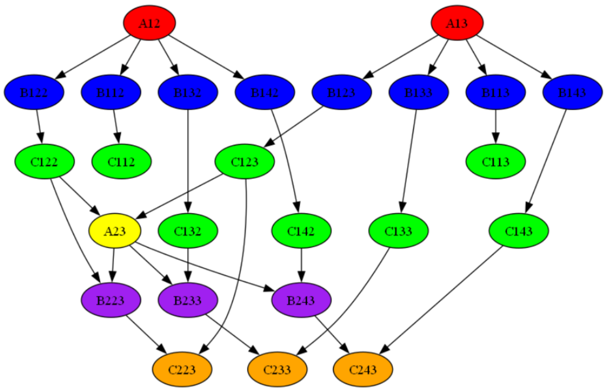

Figure 2: Enter Caption

Jak widać w naszym grafie występują nadmiarowe krawędzie - nie jest to niepoprawne z punktu widzenia teoretycznego ale nie chcemy tych krawędzi na rysunku. W kodzie stworzymy więc dodatkowe warunki eliminujące te krawędzie. Po wprowadzeniu naszych zmian graf wygląda następująco:

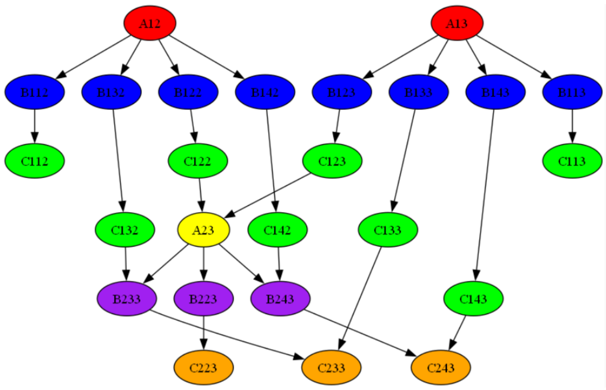

Figure 3: Enter Caption

2.5 Postać normalna Foaty

Na naszym grafie kolejne warstwy FNF są wyróznione różnymi kolorami i możemy łatwo zauważyć, że przyjmując r = 1,2,...,n-1 warstwę FNF możemy zbudować w następujący sposób:

$$F_{A,r} = \{A_{r,k} \mid r < k <= n\}$$

$$F_{B,r} = \{B_{r,j,k} \mid r < j <= n+1 \land r < k <= n\}$$

$$F_{C,r} = \{C_{r,j,k} \mid r < j <= n+1 \land r < k <= n\}$$

Wtedy całe FNF wygląda w następujący sposób:

$$FNF = [F_{A,1}][F_{B,1}][F_{C,1}][F_{A,2}][F_{B,2}][F_{C,2}]...[F_{A,n-1}][F_{B,n-1}][F_{C,n-1}]$$

3 Kod i program

3.1 Pobranie macierzy wejściowej z pliku i zapisanie wynikowej

Listing 1: Pobranie macierzy

```python
#tworzenie macierzy z pliku
def create_matrix_from_file(file_path):
    try:
        with open(file_path, 'r') as file:
            lines = file.readlines()

        num_rows = int(lines[0].strip())

        matrix = []
        x_vector = list(map(float, lines[-1].strip().split()))
        i = 0
        for line in lines[1:-1]:
            row = list(map(float, line.strip().split()))
            row.append(x_vector[i])
            i += 1

            matrix.append(row)

        return matrix

    except Exception as e:
        print(f"Error while reading from file: {e}")
        return []
```

Listing 2: Zapisanie macierzy

```python
#zapisywanie macierzy wynikowej do pliku
def save_matrix_to_file(file_path,matrix):
    try:
        with open(file_path,'w') as file:
            file.write(f"{len(matrix)}\n")

            for row in matrix:
                file.write(" ".join(map(str, row[:-1])) + "\n")

            x_vector = [round(row[-1],10) for row in matrix]
            file.write(" ".join(map(str, x_vector)) + "\n")

    except Exception as e:
        print(f"Error while saving to file: {e}")
    return
```

3.2 Stworzenie alfabetu Σ

Funkcje `create_sigma` oraz `gauss`$_{i}$$teration mogybyzostauytedowykonaniaeliminacjiGauss'a.Stdzakon$

Listing 3: Tworzenie alfabetu

```
def m_idx(i):
    return i + 1

#jedna iteracja w algorytmie gauss'a - czyli wyzerowanie elementu M(i,k)
#zgodnie z nasza teoria to ciag operacji s(i,k)
#tworzymy alfabet sigma
def gauss_iteration(M,i,k,sigma):
    cols = len(M[0])

    #operacja A
    # m_k_i = M[k][i] / M[i][i]
    sigma.append(('A', m_idx(i), m_idx(k)))

    for j in range(i,cols):
        #operacja B
        # n_k_i = M[i][j] * m_k_i
        sigma.append(('B', m_idx(i), m_idx(j), m_idx(k)))

        #operacja C
        # M[k][j] -= n_k_i
        sigma.append(('C', m_idx(i), m_idx(j), m_idx(k)))

    return

#caly gauss czyli ciag wszystkich podciagow p(i,k)
def create_sigma(M):
    rows = len(M)

    sigma = []

    for i in range(rows-1):
        for k in range(i+1,rows):
            gauss_iteration(M,i,k,sigma)

    return sigma
```

3.3 Stworzenie relacji zależności

Funkcje create_sigma sa odpowiedzialna za dodatkowe warunki odpowiadajace za usuwanie nadmiarowych krawędzi

Listing 4: Tworzenie D

```python
#funkcje pomocnicze do tworzenia zaleznosci
def d1(A,B):
    return A[0] == 'A' and B[0] == 'B' and A[1] == B[1] and A[2] == B[3]

def d2(B,C):
    return B[0] == 'B' and C[0] == 'C' and B[1] == C[1] and B[2] == C[2] and B[3] == C[3]

def d3(C,A):
    return C[0] == 'C' and A[0] == 'A' and C[2] == A[1] and (C[3] == A[1] or C[3] == A[2])

def d4(C,B):
    return C[0] == 'C' and B[0] ==  'B' and C[2] == B[2] and B[1] == C[3]

def d5(C1,C2):
    return C1[0] == 'C' and C2[0] == 'C' and  C1[2] == C2[2] and C1[3] == C2[3]

#dodatkowe warunki zapobiegajace nadmiarowym krawedziom
#wyznaczone empirycznie - patrzac na graf Dickerta
def d3_prim(C,A):
    return C[1] == A[1] - 1

def d4_prim(C,B):
    return C[1] == B[1] - 1 and B[2] != B[1]

def d5_prim(C1,C2):
    return C1[1] == C2[1] - 1 and C2[1] != C2[2]

def create_D(sigma):
    e = len(sigma)
    D = []
    D_prim = []
    I = []
    for i in range(e):
        for j in range(i+1,e):
            operation1 = sigma[i]
            operation2 = sigma[j]

            flag = True

            if d1(operation1,operation2):
                D.append([operation1,operation2])
                D_prim.append([operation1,operation2])
                flag = False

            if d2(operation1,operation2):
                D.append([operation1,operation2])
                D_prim.append([operation1,operation2])
```

```
                flag = False

            if d3(operation1,operation2):
                D.append([operation1,operation2])
                if d3_prim(operation1,operation2):
                    D_prim.append([operation1,operation2])
                flag = False

            if d4(operation1,operation2):
                D.append([operation1,operation2])
                if d4_prim(operation1,operation2):
                    D_prim.append([operation1,operation2])
                flag = False

            if d5(operation1,operation2):
                D.append([operation1,operation2])
                if d5_prim(operation1,operation2):
                    D_prim.append([operation1,operation2])
                flag = False

            if flag:
                I.append([operation1, operation2])

    return D_prim,D,I
```

3.4 Tworzenie FNF oraz Grafu Diekert'a

Listing 5: Funkcje pomocnicze

```
#znajdowanie pierwszej warstwy - czyli warstwy wszystkich wierzcholkow ktore nie maja parentow
def find_first_fnf_layer(edges):

    def has_parent(node):
        for u,v in edges:
            if v == node:
                return True
        return False

    layer = []
    for u,v in edges:
        if not has_parent(u) and u not in layer:
            layer.append(u)

    return layer

#znaleziennie wszystkich wierzcholkow w grafie
def find_all_nodes(edges):
    nodes = []
    for u,v in edges:
        if u not in nodes:
            nodes.append(u)
        if v not in nodes:
            nodes.append(v)

    return nodes

#funkcja pomocnicza usuwajaca wczesniej zdefiniowane wierzcholki koncowe
def remove_last_edges(edges):
    to_del = []
    for u,v in edges:
        if not v:
            to_del.append([u,v])

    for u,v in to_del:
        edges.remove([u,v])
```

Mając już relację D bez powtarzających się krawędzi, mamy tak naprawdę wszystkie krawędzie naszego grafu. Wykorzystujemy je więc do stworzenia FNF w następujący sposób:

1. Znajdujemy pierwszą warstwę grafu, używając wcześniej zdefiniowanej funkcji find_first_fnf_layer.
2. Dodajemy pierwszą warstwę do FNF.
3. Usuwamy pierwszą warstwę z grafu.
4. Ponownie znajdujemy pierwszą warstwę.
5. Powtarzamy kroki, dopóki w grafie pozostają krawędzie.

Listing 6: Stworzenie FNF

```python
def find_FNF(edges):
    #funkcja pomocnicza do dodania krawedzi (ostatni_node, None)
    def add_last_edges():
        for node in nodes:
            flag = True
            for u,v in edges:
                if node == u:
                    flag = False
            if flag:
                edges.append([node,None])

    fnf = []
    nodes = find_all_nodes(edges)
    add_last_edges()
    #algorytm znajdowania fnf z grafu
    #bierzemy pierwsza warstwe
    current_fnf_layer = find_first_fnf_layer(edges)

    #dopoki mamy warstwe
    while current_fnf_layer:
        #dodajemy warstwe do fnf
        fnf.append(current_fnf_layer)

        #usuwamy krawedzie pomiedzy wezlami naszej warstwy a kolejnymi
        edges = [(u, v) for u, v in edges if u not in current_fnf_layer]

        #znajdujemy kolejna warstwe - przez to ze usunelismy krawedzie, mozemy wykorzystac
        #znajdowanie wierzcholkow bez parentow
        current_fnf_layer = find_first_fnf_layer(edges)

    return fnf
```

Mając krawędzi oraz klasy Foaty możemy stworzyć Graf z odpowiednim kolorowaniem wierzchołków per klasa.

Listing 7: Rysowanie Grafu

```python
#rysowanie grafu dickerta
def draw_dickert_graph_with_layers(edges, layers):

    def create_label(node):
        return f"{node[0]}{''.join(map(str, node[1:]))}"

    remove_last_edges(edges)

    colors = ["red", "blue", "green", "yellow", "purple", "orange", "pink", "cyan","grey"]

    graph = Digraph(format='png')

    #kolorowanie wezlow
    for i, layer in enumerate(layers):
        #kazda warstwa ma inny kolor
        color = colors[i % len(colors)]
        for node in layer:
```

```python
            label = create_label(node)
            graph.node(label, style="filled", fillcolor=color)

    #narysowanie krawedzi
    for edge in edges:
        start = create_label(edge[0])
        end = create_label(edge[1])
        graph.edge(start, end)

    graph.render("dickert_graph_colored", view=True)
```

3.5 Scheduler

Listing 8: Scheduler

```python
from concurrent.futures import ThreadPoolExecutor

#wykonanie danej operacji
def execute_operation(op, matrix,results):
    if op[0] == 'A':
        i, k = op[1]-1, op[2]-1

        results[('A',i,k)] =  matrix[k][i] / matrix[i][i]

    elif op[0] == 'B':
        i, j, k = op[1]-1, op[2]-1, op[3]-1
        m_k_i = results[('A',i,k)]

        results[('B',i,j,k)] = matrix[i][j] * m_k_i

    elif op[0] == 'C':
        i, j, k = op[1]-1, op[2]-1, op[3]-1
        n_k_i = results[('B',i,j,k)]

        matrix[k][j] -= n_k_i
    else:
        raise ValueError("Nieznana operacja")
    return


#wykonanie alg. eliminacji gaussa po uwzglednieniu klas Foaty
#wszystkie operacje w danej klasie mozemy wykona rownolegle
def scheduler(fnf, matrix):

    results = {}

    for layer in fnf:
        #zarzadzanie pula watkow
        with ThreadPoolExecutor() as executor:
            #executor.submit - odpala w nowym watku
            futures = [executor.submit(execute_operation, op, matrix,
               results) for op in layer]

            #synchronizacja
            for future in futures:
                future.result()
    return matrix
```

3.6 Uruchomienie oraz wyniki

Listing 9: Wykonanie programu

```
def create_ident_matrix(M):
    rows = len(M)
    cols = len(M[0])

    for i in range(rows):
        # normalizacja wiersza zeby glowny element byl rowny 1
        divisor = M[i][i]
        for k in range(cols):
            M[i][k] /= divisor

        # wyzerowanie wszystkich innych elementow
        for j in range(rows):
            if i != j:
                factor = M[j][i]
                for k in range(cols):
                    M[j][k] -= factor * M[i][k]

    return matrix

file_path = 'ex3.txt'

matrix = create_matrix_from_file(os.path.join("examples",file_path))
sigma = create_sigma(matrix)

print(f'Alfabet: {sigma}\n')

edges,D,I = create_D(sigma)

print(f'Relacja zaleznosci: {D}\n')
print(f'Postac normalna Foaty:')

fnf = find_FNF(edges)

for layer in fnf:
    print(layer)

draw_dickert_graph_with_layers(edges,fnf)

gauss = scheduler(fnf,matrix)

matrix_ident = create_ident_matrix(gauss)

save_matrix_to_file(os.path.join("results",file_path),matrix_ident)
```

Wyniki:

```
Alfabet: [('A', 1, 2), ('B', 1, 1, 2), ('C', 1, 1, 2), ('B', 1, 2, 2), ('C', 1, 2, 2), ('B', 1, 3, 2), ('C', 1, 3, 2), ('B', 1
, 4, 2), ('C', 1, 4, 2), ('A', 1, 3), ('B', 1, 1, 3), ('C', 1, 1, 3), ('B', 1, 2, 3), ('C', 1, 2, 3), ('B', 1, 3, 3), ('C', 1,
 3, 3), ('B', 1, 4, 3), ('C', 1, 4, 3), ('A', 2, 3), ('B', 2, 2, 3), ('C', 2, 2, 3), ('B', 2, 3, 3), ('C', 2, 3, 3), ('B', 2,
4, 3), ('C', 2, 4, 3)]

Relacja zaleznosci: [[('A', 1, 2), ('B', 1, 1, 2)], [('A', 1, 2), ('B', 1, 2, 2)], [('A', 1, 2), ('B', 1, 3, 2)], [('A', 1, 2)
, ('B', 1, 4, 2)], [('B', 1, 1, 2), ('C', 1, 1, 2)], [('B', 1, 2, 2), ('C', 1, 2, 2)], [('C', 1, 2, 2), ('A', 2, 3)], [('C', 1
, 2, 2), ('B', 2, 2, 3)], [('B', 1, 3, 2), ('C', 1, 3, 2)], [('C', 1, 3, 2), ('B', 2, 3, 3)], [('B', 1, 4, 2), ('C', 1, 4, 2)]
, [('C', 1, 4, 2), ('B', 2, 4, 3)], [('A', 1, 3), ('B', 1, 1, 3)], [('A', 1, 3), ('B', 1, 2, 3)], [('A', 1, 3), ('B', 1, 3, 3)
], [('A', 1, 3), ('B', 1, 4, 3)], [('B', 1, 1, 3), ('C', 1, 1, 3)], [('B', 1, 2, 3), ('C', 1, 2, 3)], [('C', 1, 2, 3), ('A', 2
, 3)], [('C', 1, 2, 3), ('C', 2, 2, 3)], [('B', 1, 3, 3), ('C', 1, 3, 3)], [('C', 1, 3, 3), ('C', 2, 3, 3)], [('B', 1, 4, 3),
('C', 1, 4, 3)], [('C', 1, 4, 3), ('C', 2, 4, 3)], [('A', 2, 3), ('B', 2, 2, 3)], [('A', 2, 3), ('B', 2, 3, 3)], [('A', 2, 3),
 ('B', 2, 4, 3)], [('B', 2, 2, 3), ('C', 2, 2, 3)], [('B', 2, 3, 3), ('C', 2, 3, 3)], [('B', 2, 4, 3), ('C', 2, 4, 3)]]

Postac normalna Foaty:
[('A', 1, 2), ('A', 1, 3)]
[('B', 1, 1, 2), ('B', 1, 2, 2), ('B', 1, 3, 2), ('B', 1, 4, 2), ('B', 1, 1, 3), ('B', 1, 2, 3), ('B', 1, 3, 3), ('B', 1, 4, 3
)]
[('C', 1, 2, 2), ('C', 1, 3, 2), ('C', 1, 4, 2), ('C', 1, 2, 3), ('C', 1, 3, 3), ('C', 1, 4, 3), ('C', 1, 1, 2), ('C', 1, 1, 3
)]
[('A', 2, 3)]
[('B', 2, 2, 3), ('B', 2, 3, 3), ('B', 2, 4, 3)]
[('C', 2, 3, 3), ('C', 2, 4, 3), ('C', 2, 2, 3)]
```

Figure 4: Enter Caption

```
3
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 1.0 1.0
```

Figure 5: Enter Caption