Ślady - zwielowątkowienie eliminacji Gaussa

- Radosław Kawa

1. Wstęp

Zadanie polega na zaprojektowaniu współbieżnego algorytmu eliminacji Gaussa. Problem rozpatrywany będzie dla macierzy współczynników M o rozmiarze $N \times N$, N-elementowego wektora wyrazów wolnych y oraz N-elementowego wektora niewiadomych x:

$$M \cdot x = y$$

Dla $N = 3$, problem wygląda następująco:

$$\begin{pmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

2. Niepodzielne czynności

Zdefiniowano następujące niepodzielne czynności wykonywane przez algorytm:

- $A_{i,k}$ – **znalezienie** $m_{k,i}$ – mnożnika dla wiersza i-tego, potrzebnego do odejmowania tego wiersza od wiersza k-tego.

$$m_{k,i} = M_{k,i} / M_{i,i}$$

- $B_{i,j,k}$ – **znalezienie** $n_{k,i,j}$ – j-tego elementu wiersza i-tego pomnożonego przez $m_{k,i}$.

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

- $C_{i,j,k}$ – odjęcie $n_{k,i,j}$ od j-tego elementu wiersza k-tego.

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

3. Alfabet w sensie teorii śladów

Zdefiniujmy najpierw podzbiory alfabetu zawierające czynności tego samego typu.

$$\Sigma_A = \{A_{i,j} \mid 1 \le i < j \le N\}$$

$$\Sigma_B = \{B_{i,j,k} \mid 1 \le i < N,\ i \le j \le N+1,\ i < k \le N\}$$

$$\Sigma_C = \{C_{i,j,k} \mid 1 \le i < N,\ i \le j \le N+1,\ i < k \le N\}$$

Alfabet w sensie teorii śladów jest sumą powyższych zbiorów

$$\Sigma = \Sigma_A \cup \Sigma_B \cup \Sigma_C$$

4. Relacja niezależności i zależności

Najpierw zostanie wyznaczona relacja zależności D. Ponownie warto podzielić ją na kilka podzbiorów zawierających elementy tylko bezpośrednio zależne.

$$D_1 = \{(A_{i,j}, B_{l,m,n}) \in \Sigma_A \times \Sigma_B \mid i = l,\ j = n\}$$

$$D_2 = \{(B_{i,j,k}, C_{l,m,n}) \in \Sigma_B \times \Sigma_C \mid i = l,\ j = m,\ k = n\}$$

$$D_3 = \{(C_{i,j,k}, A_{l,m}) \in \Sigma_C \times \Sigma_A \mid i = l - 1,\ j = l,\ (k = l \vee k = m)\}$$

$$D_4 = \{(C_{i,j,k}, B_{l,m,n}) \in \Sigma_C \times \Sigma_B \mid l \ne m,\ i = l - 1,\ j = m,\ k = l\}$$

$$D_5 = \{(C_{i,j,k}, C_{l,m,n}) \in \Sigma_C \times \Sigma_C \mid l \ne m,\ i = l - 1,\ j = m,\ k = n\}$$

Relację zależności D można wyrazić wzorem:

$$D = sym((D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5)^+) \cup I_A$$

Relację niezależności I wygląda następująco:

$$I = \Sigma^2 - D$$

5. Algorytm eliminacji Gaussa w postaci ciągu symboli alfabetu

Niech $O_{i,k}$ oznacza odjęcie k-tego wiersza od wiersza i-tego z pominięciem elementów na kolumnach j-tych, gdzie $j < i$. Kolumny te powinny już być odpowiednio wyzerowane.

$$O_{i,k} = \left(A_{i,k}, B_{i,i,k}, C_{i,i,k}, B_{i,i+1,k}, C_{i,i+1,k}, \dots, B_{i,N+1,k}, C_{i,N+1,k}\right)$$

Niech Z_i oznacza wyzerowanie wartości w i-tej kolumnie pod przekątną (może skutkować przekształceniami także poza i-tą kolumną.

$$Z_i = \left(O_{i,i+1}, O_{i,i+2}, \dots, O_{i,N}\right)$$

Wówczas algorytm eliminacji Gaussa możemy przedstawić jako ciąg:

$$G_N = (Z_1, Z_2, \dots, Z_{N-1})$$

Przykładowo, dla $N = 3$, algorytm wygląda następująco:

$$\begin{aligned} G_3 &= (Z_1, Z_2) \\ &= \left(O_{1,2}, O_{1,3}, O_{2,3}\right) \end{aligned}$$

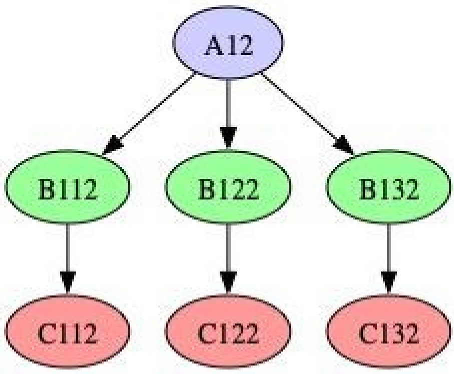
$$\begin{aligned} O_{1,2} &= \left(A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2}\right) \\ O_{1,3} &= \left(A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3}\right) \\ O_{2,3} &= \left(A_{2,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3}\right) \end{aligned}$$

6. Graf Diekerta

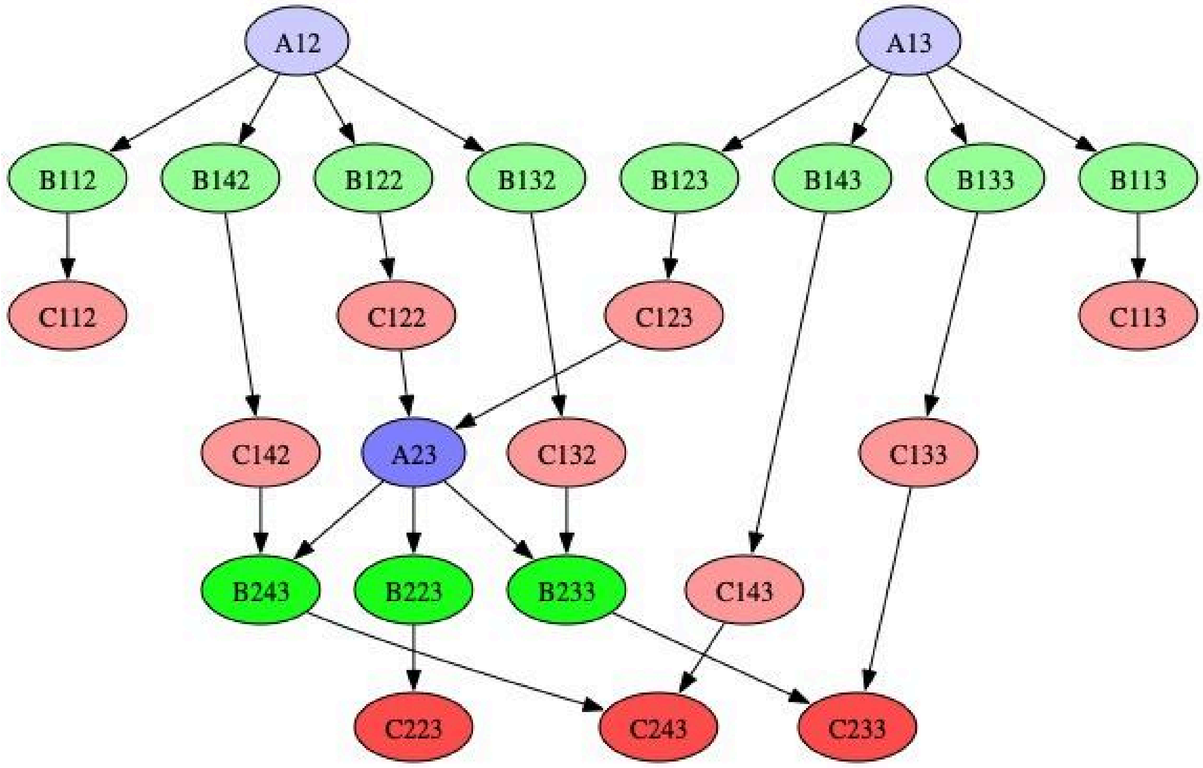
Podzbiory $D_1, \ldots, D_5$ zostały wyznaczone w taki sposób, by były ze sobą rozłączne. Oznacza to, że zawierają tylko zależności bezpośrednie. Oprócz tego, skierowane są zgodnie z postępowaniem algorytmu. Wobec tego, zbiory te wyznaczają krawędzie w grafie Diekerta:

$$E = D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5$$

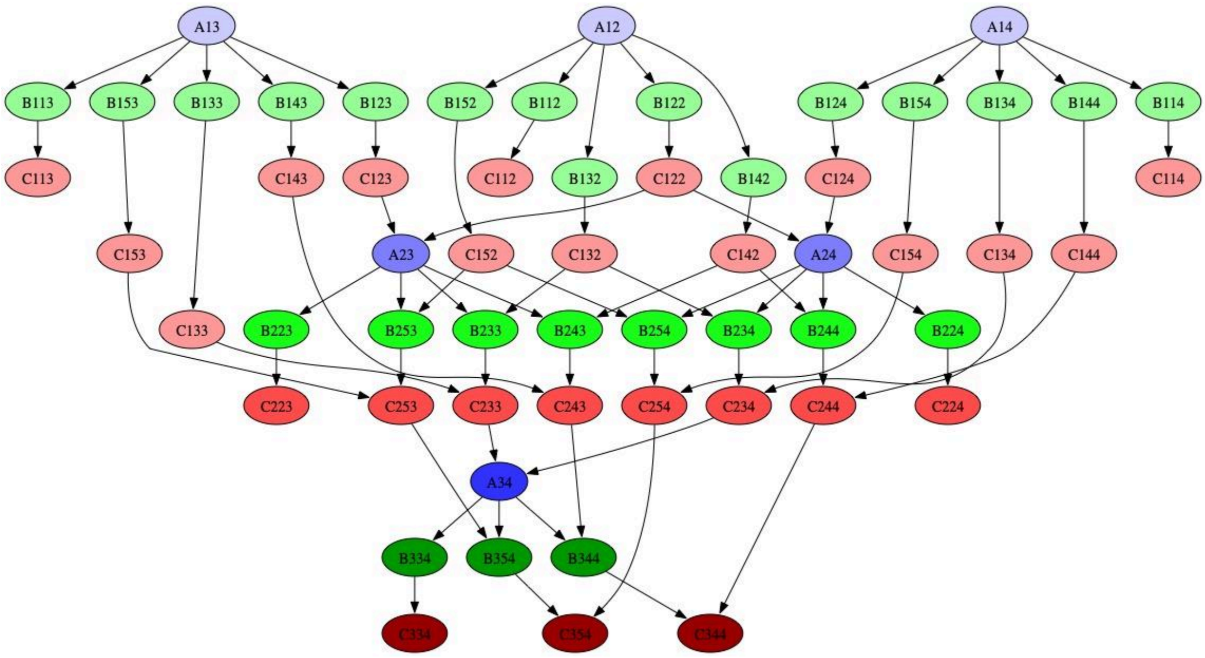
W języku Python napisano program (gauss_diekert.py) generujący graf Diekerta dla zadanego N. Wykorzystano pakiet graphviz. Graf generowany jest w formacie dot. Poniżej przedstawiono wygenerowane grafy dla N = 2, N = 3 i N = 4. Grafiki zostały wygenerowane za pomocą Graphviz Online.



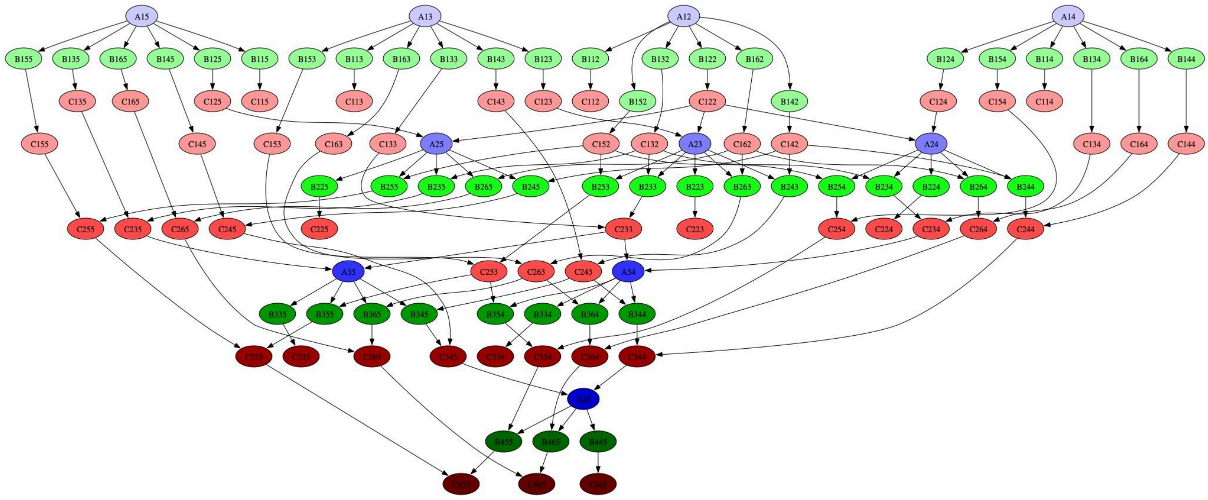
Rys. 1. Graf Diekerta współbieżnego algorytmu Gaussa dla N = 2. Kolorami oznaczono poszczególne klasy Foaty



Rys. 2. Graf Diekerta współbieżnego algorytmu Gaussa dla N = 3. Kolorami oznaczono poszczególne klasy Foaty.



Rys. 3. Graf Diekerta współbieżnego algorytmu Gaussa dla N = 4. Kolorami oznaczono poszczególne klasy Foaty.



Rys. 4. Graf Diekerta współbieżnego algorytmu Gaussa dla N = 5. Kolorami oznaczono poszczególne klasy Foaty.

7. Postać normalna Foaty

Na powyższych rysunkach można łatwo zaobserwować podział na poszczególne klasy Foaty. Podział ten zilustrowano kolorami wierzchołków. Przyjmijmy oznaczenia pomocnicze:

$$F_{A,x} = \{A_{i,j} \in \Sigma_A \mid i = x\}$$
$$F_{B,x} = \{B_{i,j,k} \in \Sigma_B \mid i = x\}$$
$$F_{C,x} = \{C_{i,j,k} \in \Sigma_C \mid i = x\}$$

Postać normalną Foaty można teraz zdefiniować następująco:

$$FNF_N = [F_{A,1}][F_{B,1}][F_{C,1}][F_{A,2}][F_{B,2}][F_{C,2}] \dots [F_{A,N-1}][F_{B,N-1}][F_{C,N-1}]$$

Przykładowo, dla N = 3, FNF wygląda następująco:

$$\begin{aligned} FNF_3 = {} & [A_{1,2}A_{1,3}][B_{1,1,2}B_{1,2,2}B_{1,3,2}B_{1,4,2}B_{1,1,3}B_{1,2,3}B_{1,3,3}B_{1,4,3}] \\ & [C_{1,1,2}C_{1,2,2}C_{1,3,2}C_{1,4,2}C_{1,1,3}C_{1,2,3}C_{1,3,3}C_{1,4,3}][A_{2,3}] \\ & [B_{2,2,3}B_{2,3,3}, B_{2,4,3}][C_{2,2,3}C_{2,3,3}, C_{2,4,3}] \end{aligned}$$

8. Implementacja

Współbieżny algorytm Gaussa zaimplementowano w języku Python. Algorytm działa zgodnie z wyprowadzoną w poprzednim punkcie postacią normalną Foaty. Różni się on jedynie indeksacją – w mojej implementacji indeksy rozpoczynają się od 0. Dodatkowo, jeżeli jest taka konieczność, wykonywany jest pivoting. Na koniec wykonywane jest podstawianie wstecz. Jako wynik uzyskiwana jest macierz wypełniona zerami, z jedynkami na przekątnej. Wektor wyrazów wolnych zawiera wartości poszczególnych niewiadomych.
Implementacja znajduje się w głównym katalogu pt. gauss.py

9. Zawartość katalogu

W głównym katalogu znajdują się następujące pliki:

- gauss_diekert.py - plik odpowiedzialny za generowanie grafu Diekerta
- matrix_input.txt – przykładowe wejście
- main.py - główny plik projektu. Wykorzystuje klasę GaussMatrix
- gauss.py – plik zawierający implementację klasy GaussMatrix. Znajduje się w niej główna funkcjonalność projektu. Metody actionA, actionB i actionC wykonują zdefiniowane wcześniej taski. Metoda concurrent_gauss wykorzystuje powyższe akcje do wykonania współbieżnej eliminacji Gaussa.
- files - katalog zawierający przykładowe wygenerowane grafy Diekerta