Obliczenia Naukowe

Lista nr 5 - Rozwiązywanie Równań Liniowych

Tomasz Niedziałek 279754

6 stycznia 2026

Spis treści

1 Wstep

Celem labolatorium było zaimplementowanie funkcji rozwiązujących liniowy układ równań postaci $Ax = b$, gdzie A jest macierzą blokową-trójdiagonalną $n \times n$ składającą się z v ($v = n/l$) bloków o wymiarach $l \times l$.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_1 & \boldsymbol{C}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{B}_2 & \boldsymbol{A}_2 & \boldsymbol{C}_2 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B}_3 & \boldsymbol{A}_3 & \boldsymbol{C}_3 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{B}_{v-2} & \boldsymbol{A}_{v-2} & \boldsymbol{C}_{v-2} & \boldsymbol{0} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B}_{v-1} & \boldsymbol{A}_{v-1} & \boldsymbol{C}_{v-1} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B}_v & \boldsymbol{A}_v \end{pmatrix},$$

Bloki zdefiniowane są następująco:

- Bloki diagonalne A_k są gęste.
- Bloki pod diagonalą B_k mają niezerowe elementy tylko w pierwszym wierszu i ostatniej kolumnie.
- Bloki nad diagonalą C_k są diagonalne

2 Struktury i złożoność pamięciowa

Dla efektywnego przechowywania macierzy zastosowano strukturę `BlockMatrix`, przechowującą bloki w wektorach, zamiast alokować pełną macierz $n \times n$. Bloki B_k, mające niezerowe elementy tylko w pierwszym wierszu i ostatniej kolumnie, zaimplementowane zostały pod postacią struktury `BorderMatrix`, przechowującej elementy niezerowe w dwóch wektorach długości l

- A_k przechowywane jako wektor v macierzy gęstych. Koszt pamięci: $v \cdot l^2 = n/l \cdot l^2 = n \cdot l$
- B_k przechowywane jako wektor $v-1$ struktur `BorderMatrix`. Koszt pamięci: $(v-1) \cdot 2l \approx 2n$
- C_k przechowywane jako wektor $v-1$ struktur `LinearAlgebra.Diagonal`, przechowującej jedynie wektor diagonalny. Koszt pamięci: $(v-1) \cdot l \approx n$

Całkowita złożoność pamięciowa struktury `BlockMatrix`:

$$M(n) \approx n \cdot l + 3n = O(n \cdot l) \underset{\text{l const}}{=} O(n)$$

3 Implementacja i złożoność czasowa

3.1 Algorytm Eliminacji Gaussa / Rozkład LU

Zastosowano wariant blokowej eliminacji Gaussa (blokowy algorytm Thomasa). Algorytm przebiega w pętli po v blokach A na diagonali.

3.1.1 Etap 1: Eliminacja w przód / Faktoryzacja

Dla każdego kroku k od 1 do $v-1$: Blok A_k jest rozkładany na czynniki L_kU_k (koszt $O(l^3)$); Obliczany jest wpływ bloku A_k na blok B_{k+1} (dopełnienie Schura). Wyznaczana jest macierz $W = A_k^{-1}C_k$. Dzięki temu, że C_k jest diagonalne, l układów równań rozwiązywane jest w czasie $O(l \cdot l^2) = O(l^3)$; Modyfikowany jest następny blok diagonalny: $A_{k+1} \leftarrow A_{k+1} - B_{k+1}W$. Dzięki rzadkiej strukturze B_{k+1}, mnożenie to kosztuje tylko $O(l^2)$.
Całkowita złożoność czasowa eliminacji:

$$T_{elim} = v \cdot (O(l^3) + O(l^3) + O(l^2)) = \frac{n}{l} \cdot O(l^3) = O(n \cdot l^2)$$

3.1.2 Etap 2: Rozwiązywanie układu

Mając faktoryzację (wariant LU) lub zmodyfikowaną macierz (wariant Gaussa):

- Forward substitution: Iteracja od $k = 1$ do v. W każdym kroku rozwiązujemy lokalny układ $L_k y_k = \ldots$ $(O(l^2))$ i odejmujemy wpływ poprzedniego bloku $(O(l))$.
- Backward substitution: Iteracja od $k = v$ do 1. W każdym kroku rozwiązujemy lokalny układ $U_k x_k = \ldots$ $(O(l^2))$ i odejmujemy wpływ następnego bloku $(O(l))$.

Całkowita złożoność rozwiązywania:

$$T_{solve} = v \cdot O(l^2) = \frac{n}{l} \cdot O(l^2) = O(n \cdot l)$$

3.2 Warianty z wyborem elementu głównego

W wariancie z częściowym wyborem elementu głównego, zamiana wierszy odbywa się tylko wewnątrz bloków diagonalnych A_k. Nie wykonujemy zamiany wierszy między blokami, aby nie zniszczyć pasmowej struktury macierzy (co prowadziłoby do wypełnienia macierzy i wzrostu złożoności). Permutacje lokalne są zapamiętywane w wektorze permutacji dla każdego bloku i uwzględniane podczas rozwiązywania lokalnych układów równań. Złożoność asymptotyczna pozostaje taka sama $(O(n))$, dochodzi jedynie narzut na wyszukiwanie maksimum w kolumnie $(O(l^2)$ na blok).

3.3 O złożoności

Kiedy l jest stałe:

- Złożoność pamięciowa: $O(n)$
- Złożoność czasowa: $O(n)$

4 Szczegółowy opis algorytmów

4.1 Opis algorytmu rozwiązywania układu bez wyboru elementu głównego

Algorytm 1 prezentuje procedurę rozwiązywania układu $Ax = b$ dla macierzy blokowo-trójdiagonalnej in-place. Macierz A jest modyfikowana w trakcie działania algorytmu, przechowując w blokach diagonalnych A_k czynniki rozkładu LU. Wektor b jest stopniowo przekształcany w wektor rozwiązania x.

Wyjaśnienie działania

Algorytm realizuje metodę eliminacji Gaussa dostosowaną do struktury blokowej. Proces przebiega w dwóch fazach:

1. **Eliminacja w przód:** Przetwarzamy macierz blok po bloku. W kroku k dokonujemy rozkładu LU gęstego bloku diagonalnego A_k. Następnie wykorzystujemy ten rozkład do wyeliminowania bloku poddiagonalnego B_{k+1} z kolejnego równania blokowego. Operacja ta modyfikuje następny blok diagonalny A_{k+1} oraz odpowiadający mu fragment wektora prawej strony x_{k+1}. Wykorzystanie rzadkiej struktury macierzy B pozwala wykonać mnożenia macierzowe w czasie $O(l^2)$ zamiast $O(l^3)$.

Algorithm 1 Rozwiązywanie układu metodą blokowej eliminacji Gaussa (Zadanie 1a)

Require: Macierz blokowa A (bloki A_k, B_k, C_k), wektor prawych stron b, liczba bloków v
Ensure: Wektor rozwiązania x

$x \leftarrow b$ ▷ Operujemy bezpośrednio na wektorze danych
Etap 1: Eliminacja w przód
for $k \leftarrow 1$ **to** v **do**
  $A_k \leftarrow \text{LU\_Decomposition}(A_k)$ ▷ Rozkład $A_k = L_k U_k$ w miejscu
  $x_k \leftarrow L_k^{-1} x_k$ ▷ Rozwiązanie układu trójkątnego $L_k y = x_k$
  if $k < v$ **then**
    $W \leftarrow A_k^{-1} C_k$ ▷ Obliczenie macierzy pomocniczej
    $A_{k+1} \leftarrow A_{k+1} - B_{k+1} W$ ▷ Modyfikacja kolejnego bloku (dopełnienie Schura)
    $z \leftarrow U_k^{-1} x_k$ ▷ Wektor pomocniczy
    $x_{k+1} \leftarrow x_{k+1} - B_{k+1} z$ ▷ Eliminacja wpływu na wektor prawej strony
  end if
end for
Etap 2: Podstawienie wstecz
$x_v \leftarrow U_v^{-1} x_v$ ▷ Rozwiązanie dla ostatniego bloku
for $k \leftarrow v - 1$ **downto** 1 **do**
  $kor \leftarrow L_k^{-1}(C_k x_{k+1})$ ▷ Obliczenie poprawki wynikającej z C_k
  $x_k \leftarrow x_k - kor$ ▷ Aktualizacja prawej strony
  $x_k \leftarrow U_k^{-1} x_k$ ▷ Ostateczne wyliczenie x_k
end for
return x

2. **Podstawienie wstecz:** Po doprowadzeniu macierzy do postaci blokowo-górnotrójkątnej, wyznaczamy rozwiązanie idąc od ostatniego bloku v do pierwszego. W każdym kroku k aktualizujemy wektor x_k o wartości wynikające z rozwiązania dla bloku $k+1$ (poprzez macierz C_k), a następnie rozwiązujemy lokalny układ równań z macierzą górnotrójkątną U_k.

4.2 Opis algorytmu rozwiązywania układu z wyborem elementu głównego

Wariant z częściowym wyborem elementu głównego różni się od wersji podstawowej tym, że dopuszcza zamianę wierszy wewnątrz bloków diagonalnych A_k w celu poprawy stabilności numerycznej. Ponieważ nie zamieniamy wierszy pomiędzy różnymi blokami, aby nie zniszczyć pasmowej struktury macierzy, proces ten nazywamy lokalnym wyborem elementu głównego.

Wymaga to zdefiniowania pomocniczej procedury rozkładu LU, która zwraca wektor permutacji (Algorytm 2).

Algorithm 2 Lokalny rozkład LU z wyborem elementu głównego

Require: Macierz gęsta A o wymiarach $l \times l$
Ensure: Macierz A nadpisana czynnikami L i U, wektor permutacji p

$p \leftarrow [1, 2, \ldots, l]$ ▷ Inicjalizacja wektora permutacji
for $k \leftarrow 1$ **to** $l - 1$ **do**
  $m \leftarrow$ indeks elementu o max module w kolumnie k od wiersza k
  if $m \neq k$ **then**
    Zamień wiersze k i m w macierzy A
    Zamień elementy $p[k]$ i $p[m]$
  end if
  for $i \leftarrow k + 1$ **to** l **do**
    $A_{ik} \leftarrow A_{ik}/A_{kk}$ ▷ Obliczenie mnożnika
    $A_{i,k+1:l} \leftarrow A_{i,k+1:l} - A_{ik} \cdot A_{k,k+1:l}$ ▷ Eliminacja
  end for
end for
return p

Główna procedura rozwiązująca (Algorytm 3) musi przechowywać wektory permutacji p_k dla każdego bloku, aby poprawnie przekształcić wektor prawej strony zarówno w fazie eliminacji, jak i podstawienia wstecznego.

Analiza różnic

Wprowadzenie wyboru elementu głównego wprowadza następujące zmiany w stosunku do wariantu podstawowego:

1. **Lokalność permutacji:** Zamiana wierszy odbywa się wyłącznie w obrębie bloków $l \times l$. Oznacza to, że relacja $PA = LU$ jest spełniona lokalnie dla każdego bloku ($P_k A_k = L_k U_k$).

2. **Pamięć:** Wymagane jest dodatkowe $O(v \cdot l) = O(n)$ pamięci na przechowanie wektorów permutacji dla każdego bloku.

3. **Podstawienie wstecz:** W równaniu $U_k x_k = L_k^{-1} P_k (b_k - C_k x_{k+1})$ człon $P_k b_k$ został uwzględniony w etapie eliminacji. Jednak człon $C_k x_{k+1}$ nie był permutowany. Dlatego przed rozwiązaniem układu z macierzą L_k, wektor poprawki $C_k x_{k+1}$ musi zostać poddany tej samej permutacji P_k, co pierwotny blok A_k.

Algorithm 3 Rozwiązywanie układu z częściowym wyborem elementu (Zadanie 1b)

Require: Macierz blokowa A, wektor b, liczba bloków v
Ensure: Wektor rozwiązania x (zapisany w miejscu b)

$x \leftarrow b$
$perms \leftarrow$ tablica wektorów permutacji długości v
Etap 1: Eliminacja w przód
for $k \leftarrow 1$ **to** v **do**
    $(A_k, p_k) \leftarrow \text{LU\_Pivot}(A_k)$ ▷ Rozkład z pivotem (Algorytm 2)
    $perms[k] \leftarrow p_k$
    $x_k \leftarrow \text{Permutuj}(x_k, p_k)$ ▷ Zastosowanie permutacji do wektora b
    $x_k \leftarrow L_k^{-1} x_k$ ▷ Rozwiązanie $L_k y = P_k b_k$
    if $k < v$ **then**
        $W \leftarrow$ Oblicz $A_k^{-1} C_k$ z uwzgl. p_k
        $A_{k+1} \leftarrow A_{k+1} - B_{k+1} W$
        $z \leftarrow U_k^{-1} x_k$
        $x_{k+1} \leftarrow x_{k+1} - B_{k+1} z$
    end if
end for
Etap 2: Podstawienie wstecz
$x_v \leftarrow U_v^{-1} x_v$
for $k \leftarrow v - 1$ **downto** 1 **do**
    $kor \leftarrow C_k x_{k+1}$ ▷ Wektor poprawki
    $kor \leftarrow \text{Permutuj}(kor, perms[k])$ ▷ Permutujemy korekrę
    $z \leftarrow L_k^{-1} kor$ ▷ Rozwiązanie układu z L_k
    $x_k \leftarrow x_k - z$ ▷ Odjęcie poprawki od aktualnego rozwiązania
    $x_k \leftarrow U_k^{-1} x_k$
end for
return x

4.3 Opis algorytmu faktoryzacji LU macierzy blokowej

Zadanie wyznaczenia rozkładu LU macierzy blokowo-trójdiagonalnej sprowadza się do wykonania eliminacji Gaussa na poziomie bloków. Algorytm działa w miejscu, nadpisując bloki diagonalne A_k ich lokalnymi rozkładami L_kU_k oraz modyfikując kolejne bloki diagonalne zgodnie ze schematem dopełnienia Schura.

4.3.1 Wariant bez wyboru elementu głównego

Algorytm 4 przedstawia procedurę faktoryzacji. Przetwarzamy macierz od górnego lewego bloku do dolnego.

Algorithm 4 Wyznaczanie rozkładu LU bez wyboru elementu (Zadanie 2a)

Require: Macierz blokowa M (bloki A, B, C), rozmiar bloku l, liczba bloków v
Ensure: Macierz M zmodyfikowana (zawiera rozkład LU)

```
for k ← 1 to v do
    LU_Dense(A_k)                          ▷ Rozkład A_k = L_k U_k w miejscu
    if k < v then
        W ← Oblicz A_k^{-1} C_k            ▷ Rozwiązanie l układów L_k U_k W = C_k
        A_{k+1} ← A_{k+1} − B_{k+1} W
    end if
end for
```

W wyniku działania algorytmu faktoryzacji, w strukturze danych przechowywane są zmodyfikowane bloki A_k (zawierające lokalne czynniki L_k i U_k) oraz niezmienione bloki B_k i C_k. Z matematycznego punktu widzenia, globalne macierze $\mathcal{L}$ i $\mathcal{U}$ dla całej macierzy blokowej A mają następującą postać:

$$\mathcal{U} = \begin{bmatrix} U_1 & C_1 & 0 & \dots \\ 0 & U_2 & C_2 & \dots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & U_v \end{bmatrix}, \quad \mathcal{L} = \begin{bmatrix} L_1 & 0 & 0 & \dots \\ \tilde{L}_2 & L_2 & 0 & \dots \\ 0 & \tilde{L}_3 & L_3 & \dots \\ \vdots & \vdots & \ddots & \ddots \end{bmatrix}$$

gdzie:

- L_k, U_k to macierze trójkątne otrzymane z lokalnego rozkładu zmodyfikowanych bloków diagonalnych, przechowywane w tablicy `M.A`.
- Bloki naddiagonalne w macierzy $\mathcal{U}$ są tożsame z macierzami C_k. Ponieważ nie ulegają one zmianie w procesie eliminacji, tablica `M.C` przechowuje gotowe fragmenty macierzy $\mathcal{U}$.
- Bloki poddiagonalne $\tilde{L}_k$ w macierzy $\mathcal{L}$ są związane z macierzami B_k zależnością:

  $$\tilde{L}_k \cdot U_{k-1} = B_k \implies \tilde{L}_k = B_k U_{k-1}^{-1}$$

  Obliczenie i zapamiętanie jawnej postaci macierzy $\tilde{L}_k$ (która jest macierzą gęstą) zwiększyłoby złożoność pamięciową algorytmu. Zamiast tego, w pamięci pozostawiamy rzadką macierz B_k (tablica `M.B`). Podczas rozwiązywania układu, operację mnożenia przez $\tilde{L}_k$ realizujemy pośrednio, wykonując sekwencję operacji na macierzach B_k i U_{k-1}, co pozwala zachować liniową złożoność pamięciową $O(n)$.

4.3.2 Wariant z częściowym wyborem elementu głównego

Wariant ten (Algorytm 5) wprowadza stabilizację numeryczną poprzez zamianę wierszy wewnątrz bloków diagonalnych.

Algorithm 5 Wyznaczanie rozkładu LU z częściowym wyborem elementu (Zadanie 2b)

Require: Macierz blokowa M, rozmiar bloku l, liczba bloków v
Ensure: Macierz M zmodyfikowana, tablica permutacji $perms$

$perms \leftarrow$ tablica o rozmiarze v
for $k \leftarrow 1$ **to** v **do**
    $(A_k, p_k) \leftarrow$ LU_Dense_Pivot(A_k) ▷ Rozkład z pivotem, zwraca permutację p_k
    $perms[k] \leftarrow p_k$ ▷ Zapamiętanie permutacji dla bloku k
    if $k < v$ **then**
        // Obliczenie W musi uwzględniać permutację p_k
        $W \leftarrow$ Oblicz $A_k^{-1}C_k$ (z użyciem p_k)
        $A_{k+1} \leftarrow A_{k+1} - B_{k+1}W$
    end if
end for
return $perms$

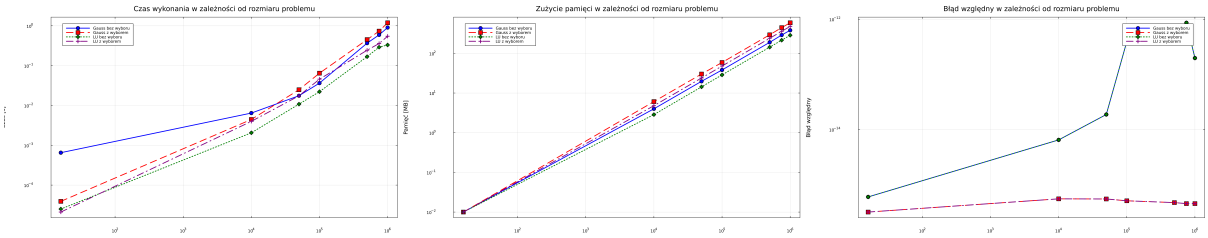
5 Wyniki i wnioski

W celu sprawdzenia działania algorytmów przeprowadziłem testy zarówno na danych zapewnionych przez profesora Zielińskiego jak i wygenerowanych przez siebie z użyciem jego modułu `matrixgen`. Testy przeprowadziłem zarówno dla rozwiązywania $Ax = b$ jak i dla $b = Ax$, gdzie $x = (1, \ldots, 1)^T$. Poniżej zamieszczona została tabela analizy złożoności $O(n^k)$ gdzie k zostało wyznaczone za pomocą regresji liniowej w skali logarytmicznej:

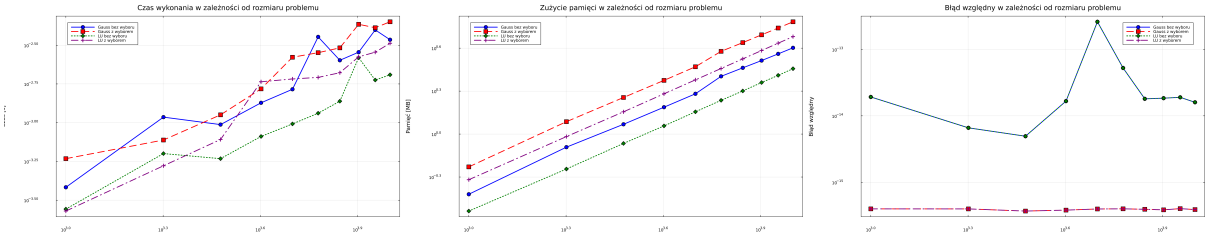
1. Jeśli złożoność to $O(n^k)$, to:
$$T(n) = c \cdot n^k$$
2. $$\log T(n) = \log c + k \cdot \log n$$
To jest równanie linii prostej: $y = a + k \cdot x$, gdzie:
    - $y = \log T(n)$ — logarytm czasu/pamięci
    - $x = \log n$ — logarytm rozmiaru problemu
    - k — nachylenie
3. Program obliczamy nachylenie prostej metodą najmniejszych kwadratów:
$$k = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
gdzie $\bar{x}$ i $\bar{y}$ to średnie wartości.
4. Nachylenie k to wykładnik w złożoności $O(n^k)$.

Tabela 1: Analiza złożoności obliczeniowej i pamięciowej algorytmów, regresja liniowa

Metoda	**Złożoność czasowa**	**Złożoność pamięciowa**
Gauss bez wyboru	$O(n^{0.65})$	$O(n^{0.96})$
Gauss z wyborem	$O(n^{0.93})$	$O(n^{0.99})$
LU bez wyboru	$O(n^{0.87})$	$O(n^{0.93})$
LU z wyborem	$O(n^{0.92})$	$O(n^{0.98})$



Rysunek 1: Zmiany czasu wykonania, zużycia pamięci oraz błędu względnego w zależności od wielkości problemu $Ax = b$ dla wszystkich 4 algorytmów



Rysunek 2: Zmiany czasu wykonania, zużycia pamięci oraz błędu względnego w zależności od wielkości problemu $b = Ax$, gdzie $x = (1, \ldots, 1)^T$ dla wszystkich 4 algorytmów

Tabela 1 potwierdza założenie liniowej złożoności.

Rysunki 1 i 2 również potwierdzają liniowość zaimplementowanych algorytmów. Ponadto zaobserwować na nich możemy również różnicę w błędzie względnym kiedy używamy wersji z wyborem elementu głównego oraz bez niego. Zgodnie z oczekiwaniami wersja z pivotem jest dokładniejsza. Widzimy również, że błąd względny zależy tylko od tego, czy algorytm korzysta z pivota czy też nie.