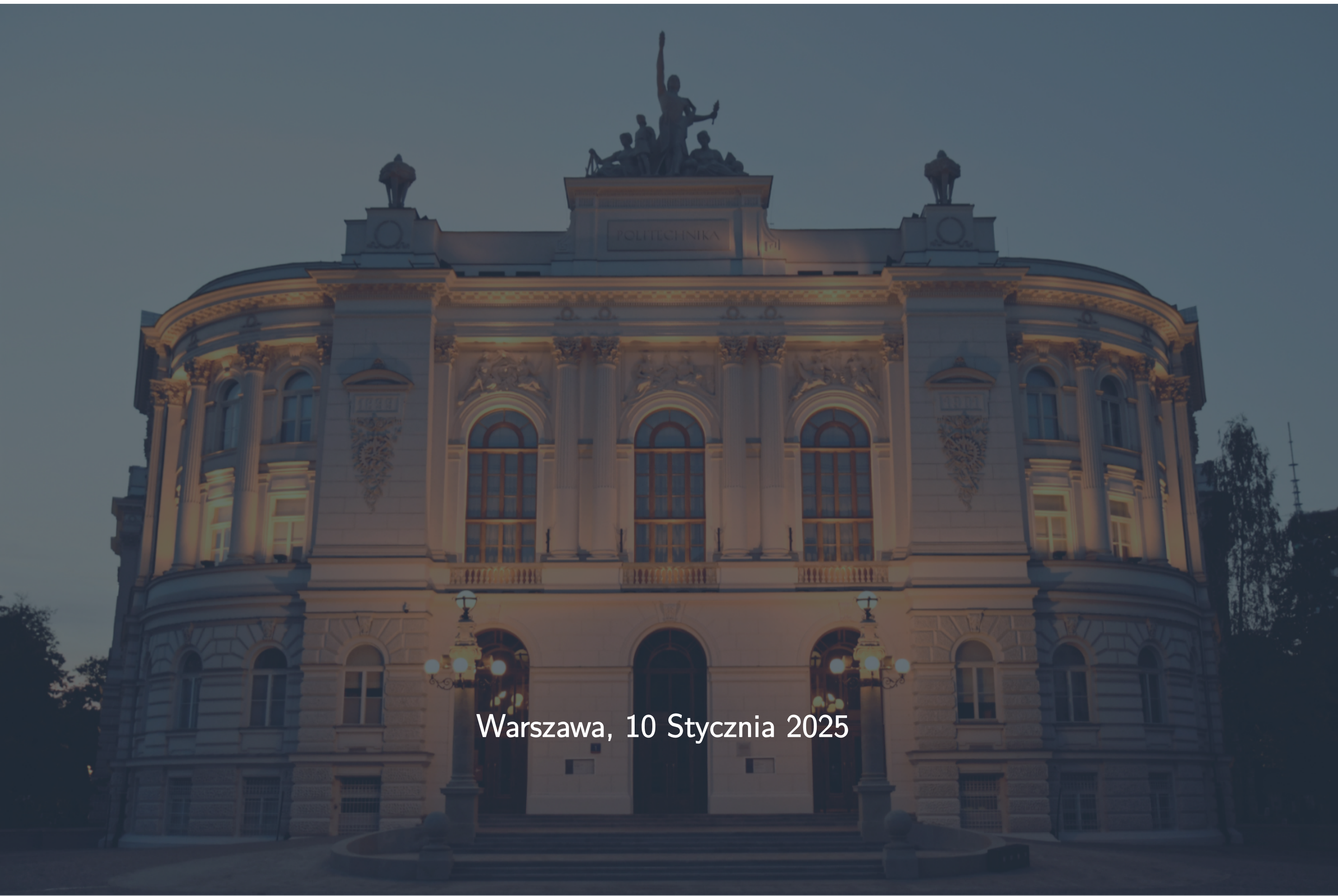
Projekt nr 2
Metody Numeryczne

Politechnika Warszawska
Wydział Matematyki
i Nauk Informacyjnych
Semestr 2024Z

Imię i Nazwisko	Numer albumu
Wiktoria Kawa	333141



Warszawa, 10 Stycznia 2025

Spis treści

1 | Wprowadzenie

W ramach 2. Projektu z przedmiotu ***Metody Numeryczne*** było zaimplementowanie za pomocą środowiska programistycznego ***MATLAB*** rozwiązania zadania danego na labolatoriach.

1.1 | Temat projektu

Celem tego projektu było wyznaczenie metody rozwiązującej układ równań liniowych $Ax = b$, gdzie

$$A = \begin{pmatrix} A_1 & 0 & 0 \\ A_2 & I & 0 \\ A_3 & A_4 & A_5 \end{pmatrix},$$

gdzie A$_i(p$ x $p)$, A_1 i A_5 są macierzami trójdiagonalnymi, symetrycznymi i dodatnio określonymi.

1.2 | Zrozumienie zadania

Zatem na samym poczatku rozpracujmy szczegóły zadania.

1.2.1 | Macierz A

Przyjrzyjmy się lepiej definicji macierzy A

- 0 - macierze z samymi zerami
- I - macierz jednostkowa, czyli 1 na przekątnej, reszta to 0
- A_1, A_5 - są macierzami trójdiagonalnymi, symetrycznymi i dodatnio określonymi
- A_2, A_3, A_4 - dowolne macierze

przy czym pamiętamy, że każda z macierzy jest wielkości p x p

1.2.2 | Układ równań

Zatem naszym zadaniem jest rozwiązanie układu równań liniowych $Ax = b$. Możemy je rozpisać:

$$\begin{bmatrix} A_1 & 0 & 0 \\ A_2 & I & 0 \\ A_3 & A_4 & A_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Po szybkim wprowadzeniu, dalszej części zajmiemy się rozwiązaniem tego problemu. Serdecznie zapraszam do lektury.

2 | Trochę matematyki

Zatem pierwsze co możemy zrobić, to wymnożyć naszą macierz A przez nasz x.

2.1 | Przemnożenie A i x

$$\begin{bmatrix} A_1 & 0 & 0 \\ A_2 & I & 0 \\ A_3 & A_4 & A_5 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} A_1 x_1 \\ A_2 x_1 + I x_2 \\ A_3 x_1 + A_4 x_2 + A_5 x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Widzimy, że otrzymujemy wstępny układ równań, z którego możemy wyznaczyć kolejne x_i.

2.1.1 | Wyznaczanie x_1

Wyznaczenie tego elementu z x jest najprostszym co nas dzisiaj spotka

$$A_1 x_1 = b_1$$
$$x_1 = A_1^{-1} b_1$$

Wiemy jednak z treści zadania, że A_1

2.1.2 | Wyznaczanie x_2

Po wyznaczeniu x_1 możemy przejść dalej:

$$A_2 x_1 + I x_2 = b_2$$
$$x_2 = b_2 - A_2 x_1$$

Przy wyznaczonym już wyżej x_1, x_2 jest wyznaczone jednoznacznie

2.1.3 | Wyznaczanie x_3

Teraz mamy wyznaczone już x_1 oraz x_2, przejdźmy zatem do wyznaczenia x_3:

$$A_3 x_1 + A_4 x_2 + A_5 x_3 = b_3$$
$$A_5 x_3 = b_3 - A_3 x_1 - A_4 x_2$$
$$x_3 = A_5^{-1}(b_3 - A_3 x_1 - A_4 x_2)$$

Przy wyznaczonym juz x_1 oraz x_2 nasze x_3 jest wyznaczone jednoznacznie, co kończy nam wyznaczanie x.

2.1.4 | Chwila na przemyślenia

Sprawdźmy jak wygląda wzór na x_2 oraz x_3 przy podstawionych wartościach na x_1 oraz x_2

- $x_2 = I(b_2 - A_2 A_1^{-1} b_1)$
- $x_3 = A_5^{-1}(b_3 - A_3 A_1^{-1} b_1 - A_4 I(b_2 - A_2 A_1^{-1} b_1)$

Personalnie, nie wiem, czy nie wstyd się do tego przyznać, ale zalewają mnie zimne poty na widok tego równania. Dlatego właśnie przy implementacji kodu będziemy wyprowadzać masze x_i po kolei.

2.2 | Różne metody i algorytmy

W tej części zastanowimy się nad własnościami naszej macierzy A. Wykorzystanie pewnych z tych własności może nam pomóc w stabilności numerycznej oraz kosztach działania funkcji.
Wiemy, że A_1 oraz A_5 są macierzami trójdiagonalnymi, symetrycznymi i dodatniookreślonymi. Pomoże nam to prawdopodobnie znaleźć kierunek w jakim możemy zmierzać.

2.2.1 | Rozkład Choleskiego

Rozkład Choleskiego jest metodą dekompozycji macierzy symetrycznej i dodatnio określonej $A \in \mathbb{R}^{n \times n}$ na iloczyn macierzy dolnotrójkątnej L oraz jej transpozycji $L^\top$, tak że:

$$A = LL^\top.$$

Rozkład ten pozwala na efektywne rozwiązywanie układów równań liniowych $Ax = b$ poprzez wykonanie następujących kroków:

1. Wyznaczenie rozkładu $A = LL^\top$ za pomocą algorytmu Choleskiego.
2. Rozwiązanie układu $Ly = b$ przy użyciu podstawienia w przód.
3. Rozwiązanie układu $L^\top x = y$ przy użyciu podstawienia wstecz.

W implementacji zastosowano funkcję `chol` dostępną w środowisku MATLAB, która zwraca macierz L w postaci dolnotrójkątnej. Kroki te zostały wykorzystane w funkcji `SolveBlockSystemCH`, gdzie rozkład Choleskiego został użyty do rozwiązywania równań dla macierzy blokowych A_1 i A_5, które są trójdiagonalne, symetryczne i dodatnio określone.

2.2.2 | Metoda Thomasa

Metoda Thomasa jest algorytmem służącym do efektywnego rozwiązywania układów równań liniowych $Ax = b$, gdzie $A \in \mathbb{R}^{n \times n}$ jest macierzą trójdiagonalną. Taka macierz posiada niezerowe elementy tylko na głównej przekątnej oraz sąsiednich przekątnych (nad- i podprzekątnej). Algorytm Thomasa to szczególny przypadek eliminacji Gaussa, zoptymalizowany dla macierzy trójdiagonalnych.
Macierz trójdiagonalna A można zapisać w postaci:

$$A = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ a_2 & d_2 & c_2 & \cdots & 0 \\ 0 & a_3 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & c_{n-1} \\ 0 & 0 & 0 & a_n & d_n \end{bmatrix},$$

gdzie a_i, d_i i c_i są odpowiednio elementami podprzekątnej, głównej przekątnej i nadprzekątnej.
Algorytm Thomasa składa się z dwóch etapów:

1. **Eliminacja w przód:** Celem eliminacji w przód jest przekształcenie układu równań w taki sposób, aby macierz stała się górnotrójkątna. Zaczynamy od pierwszego równania i kolejno eliminujemy elementy podprzekątne a_i.

   Dla $i = 2, \ldots, n$, aktualizujemy współczynniki:

   $$\tilde{d}_i = d_i - \frac{a_i c_{i-1}}{\tilde{d}_{i-1}}, \quad \tilde{b}_i = b_i - \frac{a_i \tilde{b}_{i-1}}{\tilde{d}_{i-1}},$$

   gdzie:

   $$\tilde{d}_1 = d_1, \quad \tilde{b}_1 = b_1.$$

   Po tym etapie układ ma postać:

   $$\begin{bmatrix} \tilde{d}_1 & c_1 & 0 & \cdots & 0 \\ 0 & \tilde{d}_2 & c_2 & \cdots & 0 \\ 0 & 0 & \tilde{d}_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & c_{n-1} \\ 0 & 0 & 0 & 0 & \tilde{d}_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \vdots \\ \tilde{b}_n \end{bmatrix}.$$

2. **Podstawienie wstecz:** Po eliminacji w przód układ jest rozwiązywany od ostatniego równania do pierwszego:

$$x_n = \frac{\tilde{b}_n}{\tilde{d}_n}.$$

Dla $i = n-1, n-2, \dots, 1$, rozwiązanie oblicza się jako:

$$x_i = \frac{\tilde{b}_i - c_i x_{i+1}}{\tilde{d}_i}.$$

Co ciekawe, u nas dla każdego i, $a_i = c_i$, ponieważ macierze A_1 oraz A_5 są macierzami symetrycznymi.

2.3 | Pomysł

Obie z tych metod pomagają nam przy rowiązywaniu macierzy A_1 oraz A_5. Ponieważ w treści naszego zadania nie ma określone, jak mamy rozwiązać nasz układ równań, to zrobimy przy pomocy zarówno Rozkładu Choleskiego jak i algorytmu Thomasa, jak i zwyczajnych funkcji `MATLAB`. Wówczas będzie można porównać ze sobą wyniki.

3 | Implementacja

Po przejściu przez teorię, pora przejść do implementacji. Oczywiście nie będzie tu żadnych kodów, aby nie zanudzać czytelnika.

3.1 | Funkcja: `cholesky_decomposition`

Funkcja `cholesky_decomposition` implementuje rozkład Choleskiego, który jest stosowany do rozwiązywania układu równań liniowych $Ax = b$, gdzie A jest macierzą symetryczną i dodatnio określoną.
Rozkład Choleskiego pozwala na rozbicie macierzy A na iloczyn macierzy dolnotrójkątnej L i jej transpozycji L^T, czyli:

$$A = LL^T,$$

gdzie L jest macierzą dolnotrójkątną.
Dzięki temu układ równań $Ax = b$ może zostać rozwiązany w dwóch etapach: 1. Rozwiązanie układu $Ly = b$ (eliminacja w przód). 2. Rozwiązanie układu $L^T x = y$ (eliminacja wstecz).

3.1.1 | Przyjmowane argumenty

Nasza funkcja przyjmuje macierz A oraz b, które są zdefiniowane tak jak wyżej.

3.1.2 | Opis działania

Funkcja `cholesky_decomposition` wykonuje następujące kroki:

1. Sprawdzenie poprawności wymiarów macierzy A oraz wektora b.
2. Obliczenie rozkładu Choleskiego macierzy A, czyli macierzy L, za pomocą funkcji `chol(A, 'lower')`.
3. Rozwiązanie układu $Ly = b$ za pomocą eliminacji w przód (`L  b`).
4. Rozwiązanie układu $L^T x = y$ za pomocą eliminacji wstecz (`L'  y`).

3.1.3 | Zwracany wynik

Funkcja zwraca nam rozwiązanie naszego układu równań, czyli x.

3.2 | Funkcja: `SolveBlockSystemCH`

Funkcja `SolveBlockSystemCH` rozwiązuje układ równań $Ax = b$, gdzie macierz A ma postać macierzy blokowej:

$$A = \begin{bmatrix} A_1 & 0 & 0 \\ A_2 & I & 0 \\ A_3 & A_4 & A_5 \end{bmatrix}$$

gdzie: - A_1 oraz A_5 są trójdiagonalnymi, symetrycznymi i dodatnio określonymi macierzami, - A_2, A_3, A_4 są dowolnymi macierzami, - I to macierz jednostkowa o odpowiednich wymiarach.
Funkcja dzieli wektor prawej strony b na trzy bloki: b_1, b_2, b_3, które są następnie wykorzystywane do rozwiązywania układów równań dla poszczególnych bloków macierzy A. Zastosowanie rozkładu Choleskiego umożliwia rozwiązanie układów równań dla macierzy trójdiagonalnych A_1 oraz A_5.

3.2.1 | Przyjmowane argumenty

Funkcja przyjmuje A_1, A_2, A_3, A_4, A_5 oraz b jako swoje argumenty.

3.2.2 | Opis algorytmu

Funkcja składa się z następujących kroków:

1. **Krok 1: Rozwiązanie** $A_1 \cdot x_1 = b_1$ — dla macierzy A_1, która jest trójdiagonalną, symetryczną i dodatnio określoną macierzą, wykorzystujemy rozkład Choleskiego. Zastosowanie tej metody umożliwia wyznaczenie wektora x_1.
2. **Krok 2: Rozwiązanie** $x_2 = b_2 - A_2 \cdot x_1$ — po obliczeniu x_1, rozwiązujemy układ dla x_2, modyfikując wektor b_2 o składnik $A_2 \cdot x_1$.
3. **Krok 3: Rozwiązanie** $A_5 \cdot x_3 = b_3 - A_3 \cdot x_1 - A_4 \cdot x_2$ — dla macierzy A_5, podobnie jak w kroku 1, stosujemy rozkład Choleskiego. Zmodyfikowany wektor b_3 uwzględnia składniki $A_3 \cdot x_1$ oraz $A_4 \cdot x_2$.
4. **Krok 4: Połączenie wyników** — końcowym krokiem jest połączenie wyników x_1, x_2 oraz x_3 w jeden wektor x, który jest rozwiązaniem układu równań $Ax = b$.

3.2.3 | Zwracany wynik

Zwrócony zostanie x.

3.2.4 | Założenia

Funkcja zakłada, że:

- Macierze A_1 oraz A_5 są trójdiagonalne, symetryczne i dodatnio określone, co pozwala na zastosowanie rozkładu Choleskiego.
- Macierz A ma postać blokową, a rozmiary poszczególnych bloków są odpowiednio dobrane.
- Wektor b jest zgodny z rozmiarami macierzy A.

3.3 | Funkcja: `is_triagonal`

Funkcja `is_tridiagonal` sprawdza, czy dana macierz A jest macierzą trójdiagonalną. Macierz jest trójdiagonalna, jeśli wszystkie elementy poza główną przekątną oraz sąsiednimi przekątnymi są równe zeru. W szczególności, macierz $A \in \mathbb{R}^{n \times n}$ jest trójdiagonalna, jeśli spełnia następujące warunki:

$$A = \begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ c_1 & a_2 & b_2 & \cdots & 0 \\ 0 & c_2 & a_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & b_{n-1} \\ 0 & 0 & 0 & c_{n-1} & a_n \end{bmatrix}$$

gdzie a_i, b_i oraz c_i są dowolnymi wartościami, a wszystkie pozostałe elementy są zerowe.

3.3.1 | Opis działania funkcji

Funkcja `is_tridiagonal` wykonuje następujące kroki:

1. Sprawdza, czy macierz A jest kwadratowa (tj. $n = m$).
2. Iteruje po wszystkich elementach macierzy A i sprawdza, czy elementy poza główną przekątną oraz sąsiednimi przekątnymi są zerowe.
3. Jeśli jakikolwiek element poza głównymi przekątnymi jest różny od zera, funkcja zwraca wartość `false`, oznaczając, że macierz nie jest trójdiagonalna.
4. W przeciwnym przypadku, funkcja zwraca wartość `true`, jeśli macierz spełnia warunki trójdiagonalności.

3.3.2 | Przyjmowane argumenty

Oczywiście przyjmowanym argumentem jest pewna macierz A.

3.3.3 | Zwracany wynik

Naszym zwróconym wynikiem jest wartość logiczna, która nam mówi, czy nasza macierz jest trójdiagonalna czy też nie.

3.4 | Funkcja `thomas_method`

Funkcja `thomas_method` służy do rozwiązywania układu równań $Ax = b$, gdzie macierz A jest trójdiagonalna. Jest to algorytm zaprojektowany specjalnie do macierzy trójdiagonalnych, który wykorzystuje tzw. *metodę Thomasa*, o której szczegóły były już mówione wcześniej.

3.4.1 | Problemy

1. Z tego co mi się wydaje, zaimplementowanej iteracyjnie Metody Thomasa nie da się zwektoryzować, ze względu na zależność kolejnych elementów od siebie. Zrobię wyjątek do zaprezentowania fragmentu kodu o którym mówię.

```
% Faza eliminacji wprzód
for i = 1:n denom = d(i) + a(i) * gamma(i);
gamma(i+1) = -c(i) / denom;
beta(i+1) = (b(i) - a(i) * beta(i)) / denom;
end
```

2. Dodatkowo z pełną świadomością, mimo że możemy zmniejszyć zapotrzebownaie pamięciowe kodu podając elementy z diagonali macierzy jako wektory, celowo w argumencie podamy pełną macierz, ponieważ taki właśnie argument wymaga od nas funkcja `cholesky_decomposition`. Robimy porównanie tych funkcji, więc zaimplementowanie danych funckji dla tych samych argumentów wydaje się być sensownym.

3.4.2 | Przyjmowane argumenty

Funkcja ta przyjmuje macierz A oraz b.

3.4.3 | Opis algorytmu

Funkcja zakłada, że:

- Macierz A jest trójdiagonalna, co jesteśmy w stanie sprawdzić za pomocą naszej funckji `is_triagonal`
- Wektor b ma odpowiednią długość, zgodną z wymiarami macierzy A.

Metoda Thomasa jest algorytmem eliminacji wprzód i wstecz, który pozwala rozwiązać układ równań z macierzą trójdiagonalną A. Macierz A ma postać:

$$A = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 \\ a_2 & d_2 & c_2 & \cdots & 0 \\ 0 & a_3 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n-1} & d_n \end{bmatrix}$$

gdzie: - d_i to elementy przekątnej głównej, - a_i to elementy podprzekątnej, - c_i to elementy nadprzekątnej. Algorytm Thomasa składa się z dwóch głównych etapów:

1. **Eliminacja wprzód:** W pierwszym kroku algorytmu obliczamy dwie nowe serie współczynników: γ i β, które będą wykorzystywane w dalszym rozwiązywaniu układu. Współczynniki te są wyliczane rekurencyjnie dla każdego wiersza macierzy A.

   Rozpoczynamy od obliczenia $\gamma_1 = 0$, a następnie iterujemy po wszystkich wierszach macierzy A (od $i = 2$ do $i = n$):

$$\gamma_i = \frac{c_{i-1}}{d_{i-1} + a_{i-1}\gamma_{i-1}} \quad \text{dla } i = 2, 3, \ldots, n$$

Następnie obliczamy β_i za pomocą wzoru rekurencyjnego:

$$\beta_i = \frac{b_i - a_{i-1}\beta_{i-1}}{d_i + a_{i-1}\gamma_{i-1}} \quad \text{dla } i = 1, 2, \ldots, n$$

Na końcu eliminacji wprzód uzyskujemy wektory γ oraz β.

2. **Eliminacja wstecz:** Po wykonaniu eliminacji wprzód, uzyskujemy układ równań, który możemy rozwiązać poprzez eliminację wstecz. W tej fazie obliczamy wartości zmiennych $x_1, x_2, \ldots, x_n$ na podstawie współczynników γ i β.

   Rozpoczynamy od obliczenia $x_n = \beta_n$, a następnie iterujemy wstecz, obliczając kolejne wartości $x_{n-1}, x_{n-2}, \ldots, x_1$ za pomocą wzoru:

$$x_i = \gamma_i x_{i+1} + \beta_i \quad \text{dla } i = n-1, n-2, \ldots, 1$$

Funkcja `thomas_method` rozwiązuje układ $Ax = b$, przy czym zakłada, że macierz A jest trójdiagonalna. W procesie eliminacji wprzód obliczane są dwa wektory: γ i β, które są następnie wykorzystywane w fazie eliminacji wstecz w celu uzyskania rozwiązania.

3.4.4 | Zwracany wynik

Zwrócony zostanie x.

3.5 | Funkcja: `SolveBlockSystemT`

Algorytm rozwiązuje ten układ w następujący sposób:

- **Krok 1: Rozdzielenie wektora b na bloki:** Wektor **b** jest dzielony na trzy części:

$$\mathbf{b_1} = b_1, \quad \mathbf{b_2} = b_2, \quad \mathbf{b_3} = b_3$$

- **Krok 2: Rozwiązanie dla x_1 za pomocą metody Thomasa:** Pierwszy układ równań $A_1\mathbf{x_1} = b_1$ jest rozwiązywany za pomocą metody Thomasa, ponieważ A_1 jest macierzą trójdiagonalną. Otrzymujemy rozwiązanie x_1.

- **Krok 3: Obliczenie x_2:** Po uzyskaniu x_1, obliczamy x_2 z równania:

$$A_2x_1 + x_2 = b_2 \quad \Rightarrow \quad x_2 = b_2 - A_2x_1$$

- **Krok 4: Obliczenie prawej strony dla x_3:** Aby rozwiązać układ dla x_3, obliczamy prawą stronę:

$$A_5x_3 = b_3 - A_3x_1 - A_4x_2$$

  po czym rozwiązujemy układ $A_5x_3 = b_3 - A_3x_1 - A_4x_2$ za pomocą metody Thomasa, ponieważ A_5 jest macierzą trójdiagonalną.

- **Krok 5: Połączenie wyników:** Po uzyskaniu x_1, x_2 i x_3, łączymy je w jeden wektor rozwiązania $\mathbf{x}$, który jest ostatecznym wynikiem:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

3.5.1 | Przyjmowane argumenty

Funkcja przyjmuje A_1, A_2, A_3, A_4, A_5 oraz b jako swoje argumenty.

3.5.2 | Zwracany wynik

Zostanie nam zwrócony x z treści projektu.

4 | Sprawdzenie wyników, porównanie metod

W tej sekcji zajmiemy się testowaniem zaimplementowanych metod oraz porównywaniem ich. Testy będziemy robić dla danych losowych.

Oto rozpatrywane przypadki:

1. Symetryczna, dodatniookreślona macierz o niskiej kondycji
2. Macierz o dużej kondycji
3. Macierze z perturbacją (bliskie liczby)
4. Układ o bardzo małych wartościach
5. Układ z macierzami rzadkimi
6. Układ o dużym bloku

Implementację macierzy pominiemy w tej części. Zajmijmy się zwróconymi wynikami.

Oto co po wywołaniu testów zwraca nam konsola:

Wyniki testów

Test 1/6:

x_{ch}	0.1397, 0.1702, -0.0493, 0.1010, 0.0037, 0.8532, 0.6107, 0.3046, 0.0415, 0.2945, -0.0274, 0.2127, 0.0232, 0.2259, 0.0414
x_{t}	0.1397, 0.1702, -0.0493, 0.1010, 0.0037, 0.8532, 0.6107, 0.3046, 0.0415, 0.2945, -0.0274, 0.2127, 0.0232, 0.2259, 0.0414
x_{exact}	0.1397, 0.1702, -0.0493, 0.1010, 0.0037, 0.8532, 0.6107, 0.3046, 0.0415, 0.2945, -0.0274, 0.2127, 0.0232, 0.2259, 0.0414

Test 2/6:

x_{ch}	0.0008, 0.0003, 0.0002, 0.0000, 0.0006, 0.6773, 0.0161, 0.5114, 0.2260, 0.6446, -0.0005, 0.0003, -0.0001, 0.0005, -0.0003
x_{t}	0.0008, 0.0003, 0.0002, 0.0000, 0.0006, 0.6773, 0.0161, 0.5114, 0.2260, 0.6446, -0.0005, 0.0003, -0.0001, 0.0005, -0.0003
x_{exact}	0.0008, 0.0003, 0.0002, 0.0000, 0.0006, 0.6773, 0.0161, 0.5114, 0.2260, 0.6446, -0.0005, 0.0003, -0.0001, 0.0005, -0.0003

Test 3/6:

x_{ch}	0.2328, 0.3746, 0.3292, 0.2463, 0.0859, 0.5153, 0.8273, -0.0508, 0.3370, 0.7761, 0.1554, 0.2151, 0.1838, 0.0890, 0.0396
x_{t}	0.2328, 0.3746, 0.3292, 0.2463, 0.0859, 0.5153, 0.8273, -0.0508, 0.3370, 0.7761, 0.1554, 0.2151, 0.1838, 0.0890, 0.0396
x_{exact}	0.2328, 0.3746, 0.3292, 0.2463, 0.0859, 0.5153, 0.8273, -0.0508, 0.3370, 0.7761, 0.1554, 0.2151, 0.1838, 0.0890, 0.0396

Test 4/6:

x_{ch}	5.5746×10^3, 5.8385×10^3, -0.4505, 3.1973, 5.9389, 0.0005, 0.0008, 0.0006, 0.0002, 0.0001, 0.6341, 0.0046, 0.5712, 0.9181, 0.5575
x_{t}	5.5746×10^3, 5.8385×10^3, -0.4505, 3.1973, 5.9389, 0.0005, 0.0008, 0.0006, 0.0002, 0.0001, 0.6341, 0.0046, 0.5712, 0.9181, 0.5575
x_{exact}	5.5746×10^3, 5.8385×10^3, -0.4505, 3.1973, 5.9389, 0.0005, 0.0008, 0.0006, 0.0002, 0.0001, 0.6341, 0.0046, 0.5712, 0.9181, 0.5575

Test 5/6:

x_{ch}	0.2126, 0.1493, 0.3120, 0.2770, 0.0896, 0.0821, 0.9808, 0.3710, 0.3640, 0.8090, 0.3344, 0.4082, 0.3335, 0.1887, 0.0644
x_{t}	0.2126, 0.1493, 0.3120, 0.2770, 0.0896, 0.0821, 0.9808, 0.3710, 0.3640, 0.8090, 0.3344, 0.4082, 0.3335, 0.1887, 0.0644
x_{exact}	0.2126, 0.1493, 0.3120, 0.2770, 0.0896, 0.0821, 0.9808, 0.3710, 0.3640, 0.8090, 0.3344, 0.4082, 0.3335, 0.1887, 0.0644

Test 6/6:

x_{ch}	-0.5334, 1.1837, -0.8942, 1.2324, -1.2357, 1.3782, -0.7268, 0.6954, -0.1305, 0.4596, 0.0329, 0.2575, 0.1251, 0.6174, -0.2097
x_{t}	-0.5334, 1.1837, -0.8942, 1.2324, -1.2357, 1.3782, -0.7268, 0.6954, -0.1305, 0.4596, 0.0329, 0.2575, 0.1251, 0.6174, -0.2097
x_{exact}	-0.5334, 1.1837, -0.8942, 1.2324, -1.2357, 1.3782, -0.7268, 0.6954, -0.1305, 0.4596, 0.0329, 0.2575, 0.1251, 0.6174, -0.2097

Czyli jak widzimy, nietrudno jest wysnuć wniosek, że obie zaimplementowane metody są poprawnie zaimplementowane. Każdy ze zwracanych wyników jest identyczny, co jest bardzo optymistyczne. Spójrzmy jednak nieco głębiej.

Tabela z błędami i czasami wykonania

Oto tabela zwrócona w `MATLAB`.

	TestNumber	Cond_A	ErrorRel_CH	ErrorRel_T	WspStab_CH	WspStab_T	WspPopr_CH	WspPopr_T	Time_CH	Time_T
1	1	5.8036	1.3054e-16	1.5707e-16	2.2492e-17	2.7065e-17	2.4447e-17	2.9051e-17	6.7020e-04	0.0014
2	2	1.0037e+03	1.4634e-16	1.4634e-16	1.4581e-19	1.4581e-19	1.7074e-19	2.4877e-19	2.0400e-05	1.6470e-04
3	3	9.5046	9.7490e-17	9.0470e-17	1.0257e-17	9.5186e-18	2.4090e-17	2.7296e-17	4.0060e-04	0.0011
4	4	1.2095e+04	3.7490e-17	9.0855e-17	3.0997e-21	7.5119e-21	2.8161e-20	1.5687e-20	3.8800e-05	2.8220e-04
5	5	5.7333	1.2419e-16	8.9920e-17	2.1661e-17	1.5684e-17	4.9419e-17	4.0501e-17	1.2780e-04	3.3230e-04
6	6	97.5444	5.9162e-16	2.7267e-16	6.0652e-18	2.7954e-18	3.8513e-17	1.9001e-17	2.6200e-05	6.8070e-04

Widzimy zdecydowanie bardzo dużo liczb. Analizując je, możemy spróbować dojść do pewnych wniosków.

Wnioski z badań porównujących obie funkcje

Test 1

- **Opis wyników:** Wartość wskaźnika kondycji macierzy (Cond_A $= 5.8036$) wskazuje na dobrze uwarunkowaną macierz. Zarówno względny błąd obliczeń dla metody CH (ErrorRel_CH $= 1.3054 \times 10^{-16}$) jak i metody T (ErrorRel_T $= 1.5707 \times 10^{-16}$) są bardzo małe, co świadczy o wysokiej dokładności obu metod.
- **Porównanie stabilności i poprawności:** Współczynniki stabilności (WspStab_CH $= 2.2492 \times 10^{-17}$, WspStab_T $= 2.7065 \times 10^{-17}$) oraz poprawności (WspPopr_CH $= 2.4447 \times 10^{-17}$, WspPopr_T $= 2.9051 \times 10^{-17}$) są bardzo zbliżone.

- **Czas wykonania:** Metoda CH działa szybciej (Time_CH = 0.00067 s) niż metoda T (Time_T = 0.0014 s).
- **Wniosek:** Dla dobrze uwarunkowanej macierzy obie metody są dokładne, ale metoda CH jest bardziej efektywna czasowo.

Test 2

- **Opis wyników:** Wskaźnik kondycji macierzy wynosi Cond_A = 1003.7, co wskazuje na bardziej uwarunkowaną macierz niż w Teście 1. Względne błędy dla obu metod (ErrorRel_CH = 1.4634×10^{-16}, ErrorRel_T = 1.4634×10^{-16}) pozostają bardzo niskie.
- **Porównanie stabilności i poprawności:** Współczynniki stabilności i poprawności są niemal identyczne (WspStab na poziomie 10^{-19}, WspPopr również w zakresie 10^{-19}).
- **Czas wykonania:** Metoda CH jest szybsza (Time_CH = 2.04×10^{-5} s) niż metoda T (Time_T = 0.000165 s).
- **Wniosek:** Obie metody wykazują wysoką dokładność i stabilność, ale metoda CH jest zdecydowanie szybsza.

Test 3

- **Opis wyników:** Wskaźnik kondycji wynosi Cond_A = 9.5046. Względne błędy dla obu metod są podobne i bardzo niskie (10^{-17}).
- **Porównanie stabilności i poprawności:** Stabilność i poprawność obliczeń są porównywalne (WspStab_CH = 1.0257×10^{-17}, WspStab_T = 9.5186×10^{-18}).
- **Czas wykonania:** Metoda CH działa szybciej (Time_CH = 0.0004 s) niż metoda T (Time_T = 0.00108 s).
- **Wniosek:** Obie metody są równie dokładne i stabilne, ale metoda CH jest bardziej efektywna czasowo.

Test 4

- **Opis wyników:** Wskaźnik kondycji macierzy (Cond_A = 12095) wskazuje na bardziej wymagającą macierz. Względne błędy dla obu metod pozostają bardzo niskie (10^{-17}–10^{-20}).
- **Porównanie stabilności i poprawności:** Współczynniki stabilności i poprawności wskazują na przewagę metody T w przypadku trudniejszych obliczeń (WspStab_T = 7.5119×10^{-21}, WspPopr_T = 1.5687×10^{-20}).
- **Czas wykonania:** Metoda CH (0.0000388 s) jest szybsza od metody T (Time_T = 0.000282 s).
- **Wniosek:** Metoda T jest bardziej stabilna, ale metoda CH wykazuje lepszą efektywność czasową.

Test 5

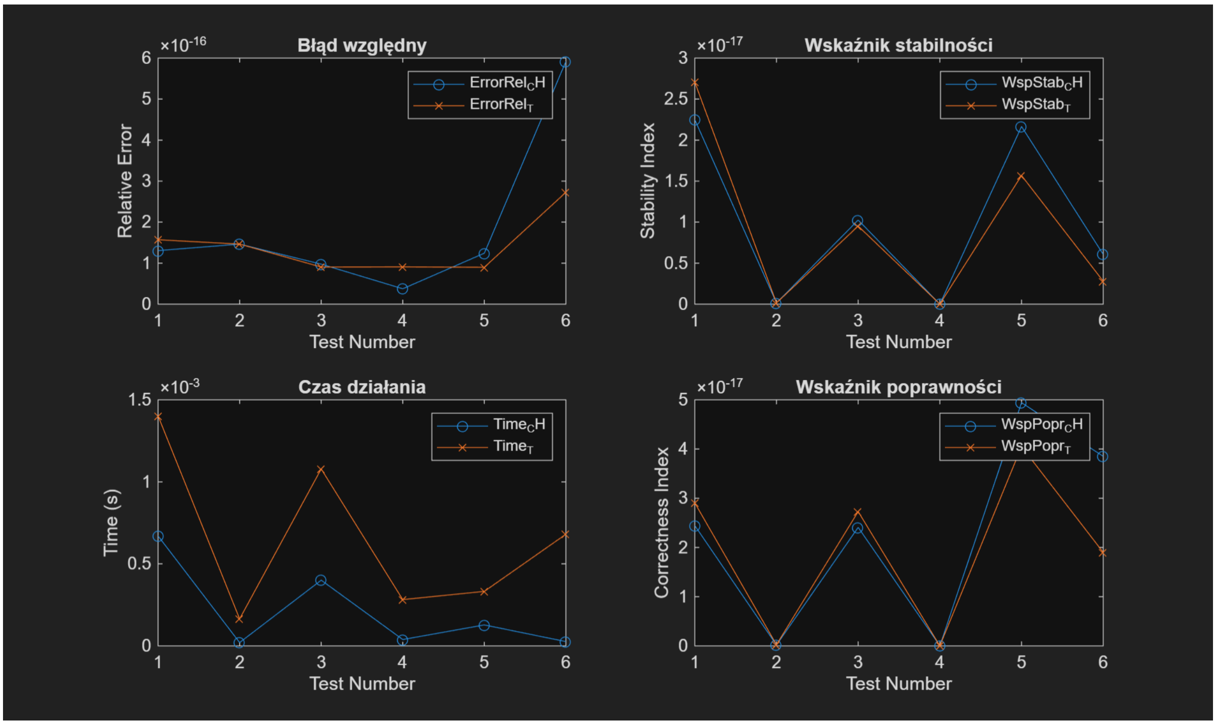
- **Opis wyników:** Kondycja macierzy wynosi Cond_A = 5.7333. Błędy względne dla obu metod są bardzo niskie (10^{-16}–10^{-17}).
- **Porównanie stabilności i poprawności:** Obie metody są stabilne i dokładne (WspStab_CH = 2.1661×10^{-17}, WspStab_T = 1.5684×10^{-17}), ale metoda CH wykazuje nieznacznie lepsze współczynniki poprawności.
- **Czas wykonania:** Metoda CH (Time_CH = 0.000128 s) działa szybciej niż metoda T (Time_T = 0.000332 s).
- **Wniosek:** Obie metody są dokładne, a metoda CH jest bardziej efektywna czasowo.

Test 6

- **Opis wyników:** Wartość Cond_A = 97.544. Względne błędy są wyższe (ErrorRel_CH = 5.9162×10^{-16}, ErrorRel_T = 2.7267×10^{-16}).
- **Porównanie stabilności i poprawności:** Stabilność metody T (WspStab_T = 2.7954×10^{-18}) jest lepsza niż metody CH (WspStab_CH = 6.0652×10^{-18}).
- **Czas wykonania:** Czas działania obu metod jest porównywalny (Time_CH = 0.0000262 s, Time_T = 0.000681 s).
- **Wniosek:** Metoda T jest bardziej stabilna i dokładna, ale czas wykonania metody CH pozostaje krótszy.

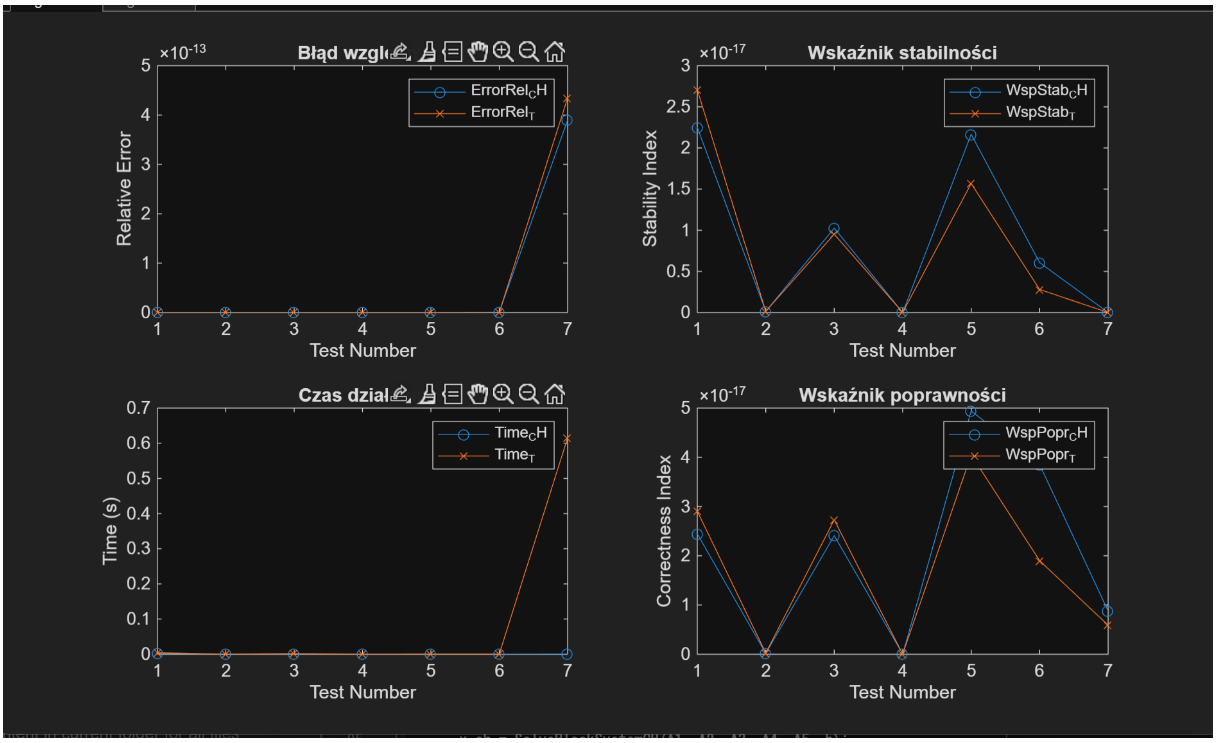
Prezentacja graficzna

Poniżej znajduje się graficzna prezentacja wyników z tabeli, żeby łatwiej było przyjrzeć się danym wynikom. Na podstawie danych wykresów jesteśmy w stanie potwierdzić nasze wnioski wyciągnięte z numerycznego porównywania wyników. Przejdźmy zatem do wyciągnięcia wniosków ogólnych.



Prezentacja Graficzna extra

Tak jak można się domyślić, przy bardzo dużych macierzach czas działania `thomas_method` znacznie może się wydłużyć ze względu na pętlę w funkcji. Ten przypadek dla dużej macierzy rozpatrujemy odrębnie, ponieważ wykresy wówczas prezentują się tak:



Oczywiście oczywistym jest też fakt, że wzrósł także błąd względny dla tak dużych macierzy, jednak jest on porównywalny dla obu tych funkcji. Skupimy się zatem na podkreśleniem tego, jak niesamowicie różni się czas działania.

Podsumowanie ogólne

- **Dokładność:** Obie metody są bardzo dokładne, ale metoda T wykazuje lepszą precyzję w przypadku trudniejszych macierzy (np. Test 6).
- **Stabilność:** Metoda T jest bardziej stabilna w przypadku macierzy o wysokim wskaźniku kondycji.
- **Czas wykonania:** Metoda CH jest ogólnie szybsza i efektywniejsza czasowo w większości przypadków.
- **Duże macierze:** Dla bardzo dużych macierzy lepiej nie korzystać z metody `thomas_method`. Jeżeli możemy, lepiej użyć jakiejś szybszej.

Rezultaty te pokazują, że wybór metody powinien zależeć od złożoności problemu i wymaganych parametrów dokładności oraz szybkości obliczeń.