Wstep

Problem polega na rozwiazywaniu układu równań liniowych zadanego jako $Ax = b$, gdzie $A \in R^{n \times n}$, $b \in R^n$, $n >= 4$ oraz A posiada szczególna postać rzadkiej macierzy kwadratowej o strukturze blokowej:

$$A = \begin{bmatrix} A_1 & C_1 & 0 & 0 & 0 & \cdots & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \cdots & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & B_{v-2} & A_{v-2} & C_{v-2} & 0 \\ 0 & 0 & \cdots & 0 & B_{v-1} & A_{v-1} & C_{v-1} \\ 0 & 0 & \cdots & 0 & 0 & B_v & A_v \end{bmatrix}$$

gdzie A_i, B_i oraz C_i sa podmacierzami kwadratowymi o rozmiarze $l = n/v$. Podmacierze maja postać:

$$B_k = \begin{bmatrix} 0 & 0 & \cdots & b^k_{1l-1} & b^k_{1l} \\ 0 & 0 & \cdots & b^k_{2l-1} & b^k_{2l} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & b^k_{ll-1} & b^k_{ll} \end{bmatrix}$$

natomiast $C_k \in R^{l \times l}$ maja postać

$$C_k = \begin{bmatrix} c^k_1 & 0 & 0 & \cdots & 0 \\ 0 & c^k_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & c^k_{l-1} & 0 \\ 0 & \dots & 0 & 0 & c^k_l \end{bmatrix}$$

Należy napisać funkcje rozwiazujaca układ Ax=b metoda eliminacji Gaussa uwzgledniajaca specyficzna postać macierzy A dla dwóch wariantów:

- bez wyboru elementu głównego
- Z cześciowym wyborem elementu głównego

Reprezentacja i przechowywanie macierzy

Macierz przechowuje w strukturze SparseMatrixCSC z pakietu SparseArrays w Julia. Pozwala to na unikniecie przechowywania dużych obszarów złożonych z samych zer. Wewnatrz struktury przechowuje sie trzy wektory, po jednym dla kolumn, rzedów i wartości. Dzieki temu formatowi możemy wydajnie iterować po wartościach niezerowych a także zaoszczedzić znaczna ilość pamieci. Wewnatrz funkcji `blockify` posługuje sie wycinkami kolumnowo-wierszowymi które kopiuje do gestych macierzy typu Matrix{Float64} w obrebie danych, które i tak

sa niezerowe w macierzy A. Dzieki temu uzyskuje małe podmacierze $l \times l$ które przetwarzam w blokowym algorytmie eliminacji Gaussa. Warto dodatkowo zaznaczyć, że tego typu reprezentacja macierzy jest szczególnie skuteczna przy macierzach blokowo–trójdiagonalnych, ponieważ wieksza cześć elementów i tak pozostaje zerowa. W efekcie minimalizujemy zbedne kopiowanie danych i skupiamy sie na rzeczywiście wystepujacych niezerowych pozycjach.

Eliminacja Gaussa

Eliminacja Gaussa to metoda rozwiażywania układó n równań z n niewiadomymi

$$A\,\mathbf{x} \;=\; \mathbf{b},$$

gdzie A jest macierza wymiaru $n \times n$, $\mathbf{x}$ – niewiadoma wektora rozmiaru n, a $\mathbf{b}$ – wektorem prawych stron (także rozmiaru n). *Eliminacja Gaussa* jest klasyczna metoda, która umożliwia przekształcenie macierzy A do postaci trójkatnej wyłacznie za pomoca dozwolonych operacji elementarnych na wierszach.

Eliminacja w przód

W pierwszym etapie dażymy do wyzerowania wszystkich elementów poniżej diagonali macierzy A. Załóżmy, że w k-tym kroku wybieramy element a_{kk} jako element główny. Nastepnie dla wszystkich wierszy $i > k$ wykonujemy:

$$\mathrm{m} \;=\; \frac{a_{ik}}{a_{kk}}, \quad (\text{wiersz } i) \;\leftarrow\; (\text{wiersz } i) \;-\; \mathrm{m} \times (\text{wiersz } k).$$

W ten sposób elementy (i, k) dla $i > k$ zostaja wyzerowane. Ta cześć procedury bywa szczególnie obciażajaca obliczeniowo, gdyż wymaga wykonania wielu operacji odejmowania i mnożenia. Jednak w przypadku macierzy o dużych obszarach zerowych operacji tych jest zdecydowanie mniej, co usprawnia cały proces.

Podstawienie wstecz

Po zakończeniu eliminacji w przód, macierz A osiaga forme:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}.$$

Z takiego układu bardzo łatwo odczytujemy wektor niewiadomych $\mathbf{x}$. W szczególności, wychodzac od ostatniego równania, mamy:

$$x_n \;=\; \frac{b_n}{a_{nn}},$$

a nastepnie, idac ,,w góre" do wiersza $i = n-1,\, n-2,\, \ldots,\, 1$:

$$x_i \;=\; \frac{1}{a_{ii}}\Big(b_i \;-\; \sum_{j=i+1}^{n} a_{ij}\, x_j\Big).$$

Na etapie podstawienia wstecz kluczowe jest to, aby diagonalne elementy macierzy A nie były zerowe badź zbyt małe, co mogłoby prowadzić do niestabilności numerycznych. Dlatego właśnie w wielu wariantach Eliminacji Gaussa wprowadza sie wybór elementu głównego.

Złożoność obliczeniowa

W najbardziej ogólnym wypadku (macierz pełna, bez dodatkowych zer) eliminacja Gaussa ma złożoność $\mathcal{O}(n^3)$. Konieczność wykonania $\sim \frac{1}{3}n^3$ operacji wynika z tego, że w k-tym kroku petli eliminacji trzeba zaktualizować elementy wierszy $i > k$ w kolumnach $j > k$, co w sumie daje kwadratowy (wzgledem n) licznik operacji w każdej z n iteracji.

Eliminacja Gaussa w macierzy blokowo–trójdiagonalnej

Moja implementacja rozwiazuje układ równań, w którym macierz A ma blokowo–trójdiagonalna strukture opisana wcześniej

- $\mathbf{A}_k$ jest *blokiem diagonalnym* (na głównej przekatnej) o rozmiarach $l \times l$.
- $\mathbf{B}_k$ jest *blokiem poddiagonalnym* (czyli ,,o jedno niżej" w macierzy), również $l \times l$.
- $\mathbf{C}_k$ jest *blokiem naddiagonalnym* (czyli ,,o jedno wyżej" w macierzy), także $l \times l$.
- Indeks k w $\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k$ zwykle przebiega od 1 do v, gdzie v to łaczna liczba bloków *na przekatnej*. Cała macierz A ma wiec wymiar $n \times n$, gdzie $n = v \cdot l$.
- $\mathbf{b}_k$ oznacza *cześć* (blok) wektora prawych stron o długości l.
- $\mathbf{x}_k$ oznacza *fragment* (blok) wektora niewiadomych, także długości l.

Z tego wzgledu $\mathbf{x}$ i $\mathbf{b}$ *dzielimy* na v podwektorów, co odpowiada podziałowi macierzy na bloki $l \times l$. Innymi słowy, w każdym kroku operujemy na macierzach i wektorach *blokowych*.

Eliminacja w przód

W standardowej (nieblokowej) eliminacji Gaussa musielibyśmy kolejno zerować elementy pod przekatna w każdej kolumnie, wykonujac operacje typu:

$$\text{wiersz}(i) \;\leftarrow\; \text{wiersz}(i) \;-\; m \;\times\; \text{wiersz}(k),$$

gdzie $m = a_{ik}/a_{kk}$.

W naszej specyficznej strukturze zamiast tego *zerujemy całe bloki* $\mathbf{B}_k$. Aby to uzyskać, definiujemy w k-tym kroku:

$$\mathbf{L}_{k+1} \;=\; \mathbf{B}_k \left(\mathbf{A}_k\right)^{-1}.$$

Dzieki tej macierzowej kombinacji ($\mathbf{B}_k$ pomnożone przez odwrotność $\mathbf{A}_k$) uzyskujemy blok, który *posłuży do wyzerowania* $\mathbf{B}_k$ w nastepnym wierszu (bloku). Nastepnie dokonujemy aktualizacji w macierzach:

$$\mathbf{A}_{k+1} \;\leftarrow\; \mathbf{A}_{k+1} \;-\; \mathbf{L}_{k+1}\,\mathbf{C}_k$$

oraz w wektorach prawych stron:

$$\mathbf{b}_{k+1} \;\leftarrow\; \mathbf{b}_{k+1} \;-\; \mathbf{L}_{k+1}\,\mathbf{b}_k.$$

W przypadku macierzy blokowej każdy taki krok pozwala zaoszczedzić wiele operacji arytmetycznych w porównaniu z metoda tradycyjna.

Podstawienie wstecz

Po $v-1$ krokach macierze $\mathbf{A}_k$ znajdujace sie na przekatnej staja sie (blokowo) górnotrójkatne, co znaczy, że sa gotowe do podstawiania wstecz. W zwykłej, nieblokowej wersji, mielibyśmy:

$$x_n = b_n/a_{nn}, \quad x_i \;=\; \frac{1}{a_{ii}}\Big(b_i - \sum_{j=i+1}^{n} a_{ij}\,x_j\Big).$$

W podejściu blokowym postepujemy analogicznie, ale na poziomie całych bloków:

$$\mathbf{x}_v \;=\; \left(\mathbf{A}_v\right)^{-1}\mathbf{b}_v,$$

$$\mathbf{x}_k \;=\; \left(\mathbf{A}_k\right)^{-1}\Big(\mathbf{b}_k \;-\; \mathbf{C}_k\,\mathbf{x}_{k+1}\Big) \quad \text{dla } k = v-1,\; v-2,\; \ldots,\; 1.$$

Każdy taki blok $\mathbf{x}_k$ jest wektorem długości l.

Wybór elementu głównego

W tradycyjnym algorytmie Gaussa, aby uniknać dzielenia przez zbyt mały (lub zerowy) element a_{kk}, stosuje sie wybór elementu głównego. W wersji blokowej analogicznie można wykonywać wybór elementu głównego z bloku

$\mathbf{A}_k$. Polega to na tym, że jeśli w blokowej macierzy $\mathbf{A}_k$ (o wymiarze $l \times l$) któraś wartość diagonalna jest bardzo bliska 0, dokonujemy zamiany wierszy *tylko w tym bloku*, a także analogicznych zamian w odpowiednich wierszach bloków $\mathbf{B}_{k+1}$ i $\mathbf{b}_k$. Cześciowy wybór elementu głównego pomaga uniknać niestabilności obliczeniowej i chroni przed nadmiernym wzrostem błedów zaokragleń. Odpowiednie dobranie wierszy w obrebie bloku potrafi znaczaco poprawić uwarunkowanie macierzy pozostałej do przetworzenia w kolejnych etapach.

Złożoność i optymalizacje

Rozmiary i liczba bloków. Zakładamy, że $\mathbf{A}_k, \mathbf{B}_k, \mathbf{C}_k$ sa macierzami $l \times l$, zaś liczba bloków na przekatnej to v. Wówczas mamy:

$$n \;=\; v \cdot l,$$

gdzie n to całkowity rozmiar macierzy A.
Koszt eliminacji. W eliminacji Gaussa $\mathcal{O}(n^3)$ wynika głównie z faktu, że w klasycznej (pełnej) macierzy musimy wyzerować duża liczbe elementów. W macierzy *blokowo–trójdiagonalnej* wiele z tych elementów *od poczatku* jest zerowych, wiec *nie musimy* wykonywać tylu operacji. W każdym kroku zerujemy *tylko* blok $\mathbf{B}_k$ i aktualizujemy $\mathbf{A}_{k+1}$.

Gdy l jest małe, a v duże, łaczna złożoność potrafi sie sprowadzić nawet do $\mathcal{O}(v)$, co oznacza $\mathcal{O}(n)$. W praktyce bywa to wielokrotnie *mniejsze* niż $\mathcal{O}(n^3)$.

Wyniki Eksperymentu

Metodologia

Przeprowadziłem porównanie czasu wykonywania Eliminacji Gaussa na danych wygenerowanych za pomoca matrixgen z paczki danych testowych. Wykonałem na każdej macierzy eliminacje trzema metodami. Metoda Tradycyjna, czyli korzystajac z operatora \z pakietu LinearAlgebra oraz dwoma własnymi.

Wnioski

Jak widać z przeprowadzonego eksperymentu, metoda przystosowana do cech macierzy z która mamy do czynienia pozwala na znaczne oszczedności w czasie obliczeniowym ponad metoda tradycyjna dostepna w pakiecie LinearAlgebra. Znajac właściwości macierzy możemy również zaoszczedzić dużo miejsca korzystajac z reprezentacji SparseArrays. Ostateczne rezultaty pokazuja, że nawet przy relatywnie niewielkich rozmiarach macierzy blokowo–trójdiagonalnych, przyrost wydajności jest wyraźny.

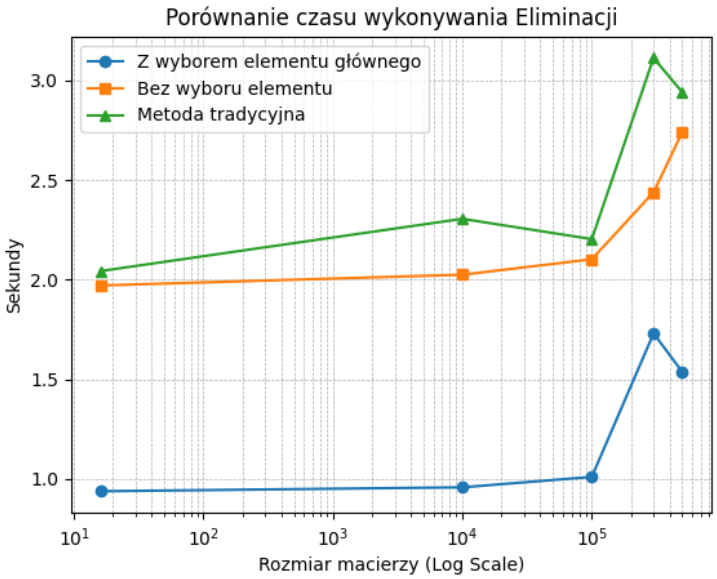

Figure 1: Wyniki eksperymentu