Obliczenia Naukowe Laboratorium 5

Adam Jamroziński

December 2024

Lista poświęcona rozwiązywaniu układów równań liniowych, dla specyficznej rzadkiej macierzy blokowej. Rozwiązaniem układu oznaczamy:

$$\mathbf{Ax} = \mathbf{b}$$

dla danych $\mathbf{A} \in \mathbb{R}^{n \times n}, \mathbf{b}^n, n \in \mathbb{N}$
Szukamy wektora $\mathbf{x} \in \mathbb{R}^n$

Rozwiązywanie układu metodą eliminacji Gaussa

Stworzymy efektywny algorytm dla specyficznej macierzy blokowej o postaci

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{C}_1 & 0 & 0 & \cdots & 0 \\ \mathbf{B}_2 & \mathbf{A}_2 & \mathbf{C}_2 & 0 & \cdots & \vdots \\ 0 & \mathbf{B}_3 & \mathbf{A}_3 & \mathbf{C}_3 & \cdots & \vdots \\ 0 & 0 & \mathbf{B}_4 & \mathbf{A}_4 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \mathbf{C}_{v-1} \\ 0 & \cdots & \cdots & \cdots & \mathbf{B}_v & \mathbf{A}_v \end{bmatrix}$$

Gdzie A, B, C to kolejno

$$\mathbf{A_k} = \begin{bmatrix} a^k_{1,1} & a^k_{1,2} & \cdots & a^k_{1,l} \\ a^k_{2,1} & a^k_{2,2} & \cdots & a^k_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ a^k_{l,1} & a^k_{l,2} & \cdots & a^k_{l,l} \end{bmatrix}$$

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

$$\mathbf{C_k} = \begin{bmatrix} c_{1,1} & 0 & 0 & \cdots & 0 \\ 0 & c_{2,2} & 0 & \cdots & 0 \\ 0 & 0 & c_{3,3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & c_{l,l} \end{bmatrix}$$

Metoda eliminacji Gaussa jest kluczowa do zrozumienia algorytmu ale nie będzie tutaj szczegółowo opisywana. Główna idea wynikająca z pseudo-kodu to doporowadzenie macierzy **A** do macierzy górnej trójkątnej poprzez zerowanie elementów poniżej diagonalnych przechodząc pokolei po pozycjach na przekątnej.

Algorithm 1: Gaussian Elimination for Solving $Ax = b$

Input: Matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$
Output: Solution vector $x \in \mathbb{R}^n$
for $k = 1$ *to* $n - 1$ **do**
  for $i = k + 1$ *to* n **do**
    $l_{i,k} = \frac{A[i,k]}{A[k,k]}$;
    for $j = k$ *to* n **do**
      $A[i, j] \leftarrow A[i, j] - l_{i,k} \cdot A[k, j]$;
    end
    $b[i] \leftarrow b[i] - l_{i,k} \cdot b[k]$;
  end
end
Backward Substitution:;
$x[n] \leftarrow \frac{b[n]}{A[n,n]}$;
for $i = n - 1$ *down to* 1 **do**
  $x[i] \leftarrow \frac{b[i] - \sum_{j=i+1}^{n} A[i,j] \cdot x[j]}{A[i,i]}$;
end
return x;

Analiza złożoności

Oczywiście oryginalny algorytm eliminacji (także ten z częściowym wyborem) działa w czasie $O(n^3)$ gdzie n to wielkość macierzy **A**. Szukając optymalizacji możemy skorzystać z rzadkiej struktury macierzy. Przechodząc po j-tej kolumnie, podczas eliminacji wartości $a_{i,j}$ znajdującej się pod pozycją $a_{j,j}$, czyli dla $j+1 \leqslant i \leqslant n$ będziemy omijali dużą część macierzy pod blokiem $B_{\lceil \frac{i}{l}+1 \rceil}$ (który sam w sobie posiada wiele wyzerowanych kolumn co wykorzystuje w implementacji, co nie jest konieczne w analizie i zostanie pominięte), z kolei gdy odejmujemy i-ty wiersz od tych stojących pod nim aby dokonać odejmowania wierszy możemy ominąc duża porcje macierzy i-tego wiersza $i > j$ i odejmować tylko niezerowe wartości j-tego wiersza, a jest ich dla każdego kroku conajwyżej $l + 1$. Wynika to z prostej analizy tj. podczas eliminacji k-tego wiersza odejmując od niego wiersz j-ty po prawej stronie pozycji (j, j) wszystkie wartości stojące za diagnonalną bloku C (po jej prawej) muszą być równe zero w przypadku braku wyboru, przypadek z wyborem omawiam niżej. To wszystko pozwala nam do optymalizacji której skutkiem jest złożonośc czasowa $O(n \cdot (l + 1) \cdot (l + 1))$, gdzie n pochodzi z każdego ciągu eliminacji wartości poniżej diagonalnej, każdy ciąg eliminacji to $(l + 1)$ operacji wynikającej z ilości wierszy po których schodzimy w dół (dla większości przypadków szybko obserwujemy, że jest to tak naprawdę mniej niż l co jest ominięte w celu łatwiejszej analizy co nie zmienia złożonośći czasowej), a finalnie $(l + 1)$ pochodzi z faktycznych odejmowań które musmy wykonać w danym wierszu korzystając z faktu powyżej. **Podsumowując i redukjąc, nasz algorytm osiąga złożoność czasową $\mathbf{O(nl^2)}$.** Warto wspomnieć że dzieki efektywnemu przechowywaniu macierzy w pamięci redukujemy maksynalnie jej rozmiar

z n^2 do, w przypadku mojej implementacji $vl^2+(v-1)l^2+(v-1)l+(v-2)l$ gdzie kolejne sumy pochodzą ze zliczenia elementów kolejnych bloków A, B, C oraz D. Co po zliczeniu daje **złożoność pamięciową zaledwie O(nl)**

Częściowy wybór w eliminacji

Kożystając z możliwości zamiany wierszy, tak aby uzyskać jaknajwiększą wartość $l_{i,j}$ zależną od $a_{j,j}$ podczas eliminacji wierszy poniżej $a_{j,j}$ napotykamy dwa problemy. Musimy uwzględnić że blok C może zawierać niezerowe wartości nad i na prawo od pozycji diagonalnych (wynikające z zamiany z wierszem który stał poniżej w bloku A). Kolejnym skutkiem częściowego wyboru jest możliwość zapisania wartości po prawo od bloku C (wynikające z zamiany z wierszem który należał do bloku B). **Uwaga, nie zmienia to złożoności czasowej algorytmu**, ponieważ zmienia to tylko stałą, którą możemy oszacować z góry, że nie wzrośnie ona więcej niż dwukrotnie. Otrzymujemy ten wynik zakładając, że teraz musimy przejść po przynajmniej $2l$ wartościach odejmując od siebie dwa różne wiersze.

Sposób przechowywania macierzy

Do reprezentacji rzedkiej macierzy **A** potrzeba jest lista bloków A, B, C, D, gdzie każdy blok jest reprezentowany przez odpowiednio A, C - zwykłe macierze rozmiaru $l \times l$.
Mimo że dla C nie jest konieczne aby przechowywać w pamięci wszystkie wartości ułatwi to implementacje eliminacji bez oraz z częściowym wyborem.
Blok B jest reprezentowany jako macierz $l \times 2$ ponieważ pusta część B jest zawsze pusta podczas rozkładu oraz podczas eliminacji Gaussa.
Pomocniczy blok D rozmiaru $2 \times l$

Potrzeba dodatkowego bloku D

Podczas eliminacji gaussa z częściowym wyborem część wierszy może zostać wymieszana co za tym idzie istnieje szansa, że macierz straci część swoich własności dotyczących tego gdzie znajdują się wartości zerowe. Jest tak w przypadku gdy musimy wymienić wiersze z dwóch różnych bloków tj. bloków A i B. W tej sytuacji sąsiedni blok C w prostej lini po prawej od danego bloku B musi zostać przesunięty na pozycje spoza bloków A, B, C. Nie jest to powód do paniki ponieważ może stać się tak tylko dla pozycji dla których po prawej istnieje element bloku A pod którym mamy niezerowe komórki bloku B. Istnieją tylko dwa rzędy takich pozycji długości l dla każdego bloku A i własnie wartości na tych pozycjach będą przechowywane w bloku D dając nam zmodyfikowaną postać macierzy

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{C}_1 & \mathbf{D}_1 & 0 & \cdots & 0 \\ \mathbf{B}_2 & \mathbf{A}_2 & \mathbf{C}_2 & \mathbf{D}_2 & \cdots & \vdots \\ 0 & \mathbf{B}_3 & \mathbf{A}_3 & \mathbf{C}_3 & \cdots & \vdots \\ 0 & 0 & \mathbf{B}_4 & \mathbf{A}_4 & \cdots & \mathbf{D}_{v-2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \mathbf{C}_{v-1} \\ 0 & \cdots & \cdots & \cdots & \mathbf{B}_v & \mathbf{A}_v \end{bmatrix}$$

Gdzie D to:

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{1,1} & d_{1,2} & d_{1,3} & \cdots & d_{1,l} \\ d_{2,1} & d_{2,2} & d_{2,3} & \cdots & d_{2,l} \end{bmatrix}$$

Testy

Testy czasowe i pamięciowe

Dla danych macierzy wielkości kolejno 16, 10000, 50000, 100000, 300000, 500000 sprawdzamy ile czasu wykonuje się algorytm eliminacji gaussa z wyborem oraz bez wyboru. Dla wszystkich danych testowych l - wielkość bloków była stała równa 4.

Wielkość macierzy (n)	**Brak wyboru (s)**	**Częściowy wybór (s)**	**Pamięć użyta**
16	0.000050	0.000047	11.703 KiB
10000	0.018615	0.022336	7.033 MiB
50000	0.136964	0.160514	34.074 MiB
100000	0.407641	0.451724	70.875 MiB
300000	2.277835	2.470890	234.078 MiB
500000	5.674225	5.925281	413.282 MiB

Tabela 1: Analiza złożoności wydajności metod eliminacji

Z tabeli możemy odczytać zarys liniowej złożonośći czasowej. Wysokie skoki pamięciowe mogą być konsekwencją ilości wykonywania głębokich kopii danych użytych w mojej implementacji.
Wzrost czasu wykonania pokrywa się z liniową złożonością czasową (dla stałego l) częściowy wybór różni się tylko czasem wybierania i przestawiania wierszy, co głónie jest zależne od ilości przestawień co z kolei zależy tylko od danego **A** na wejściu.

Błędy względne

Dla danych o tych samych rozmiarach i parametrach generujemy (mnożymy przez wektor jedynek $x = [1, 1, \cdots, 1]$) wektory prawych stron i sprawdzamy średni błąd na każdej składowej po wykonaniu eliminacji Gaussa odpowiednio z wyborem oraz bez

Wielkość macierzy (n)	**Brak wyboru (s)**	**Częściowy wybór**
16	6.8695×10^{-16}	2.5674×10^{-16}
10000	3.6280×10^{-15}	3.6636×10^{-16}
50000	5.2844×10^{-15}	3.9987×10^{-16}
100000	1.4437×10^{-14}	3.6711×10^{-16}
300000	4.3422×10^{-15}	3.8186×10^{16}
500000	4.1148×10^{-15}	3.6216×10^{-16}

Tabela 2: Testy precyzji metod eliminacji

Na tabeli opisana jest precyzyjność metody eliminacji Gaussa (optymalizacje obejmowały tylko omijanie niepotrzebnych działań arytmetycznych) zatem obie metody dają rezultat metody Gaussa taka jaką ona jest. Dzięki wyborowi dużych wartości diagnonalnych elementów przez które dzielimy, mnożniki $l_{i,j}$ możemy reprezentować z większą precyzją (mniejszymi stratami przez dzielenie przez małe wartości). Co daje nam średnio 10 razy lepszą precyzje dla danych testowych

Rozkład LU

Faktoryzacja na LU: $A = LU$

Optymalizacje w przechowaniu L **i** U wynikają z tego że obie posiadają formę macierzy blokowej **A**

- dla L wartości niezerowe znajdują się na pozycjach na których znajdują się niezerowe wartości pod przekątną macierzy **A**.
- Skłowa U to po prostu tiangulacja A z eliminacji gaussa (którą już reprezentowaliśmy blokowo).

Ogólnie:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n,1} & l_{n,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{n,n} \end{bmatrix}.$$

Zostanie skrócone o poszczególne bloki (nigdy nie wymieniamy czegoś co jest zerem z wartościom niezerową) lub w przypadku U rozszerzone o pomocnicze bloki D.
Faktoryzacja i interpretacja pozwala uprzednie wyliczenie U wraz zapamiętaniem L^{-1}, mówiącym o tym które pozycje wektora prawych stron musielibysmy poodejmować aby uzyskać wynik układu równań.
Rozwiązanie to nie różni się złożonością czasową od zwykłej (bez pamiętania zamian i odejmowań w macierzy L) eliminacji Gaussa. Ponieważ dalej macierze trzymamy w rzadkiej macierzy blokowej złożoność pamięciowa również nie ulega zmianie.
Aby rozwiązać układ równań pozostaje obliczyć wektor y =L^{-1}

$$LU\mathbf{x} = \mathbf{b}$$

$$U\mathbf{x} = L^{-1}\mathbf{b} = y$$

Co możemy łatwo osiągnąc zauważając, że

$$L^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -l_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ -l_{n,1} & -l_{n,2} & \cdots & 1 \end{bmatrix}$$

Następnie wymnażając i otrzymując **y** podobnie jak w eliminacji Gaussa otrzymujemy **x** korzystając z prostego algorytmu

Algorithm 2: Backward Substitution:

$x[n] \leftarrow \frac{b[n]}{A[n,n]}$;
for $i = n-1$ *down to* 1 **do**
  $x[i] \leftarrow \frac{b[i] - \sum_{j=i+1}^{n} A[i,j] \cdot x[j]}{A[i,i]}$;
end
return x;

Oba etapy rozwiązywania układu $LU\mathbf{x} = \mathbf{b}$ mają liniową złożoność czasową dla stałego l.