Sprawozdanie z listy 5

Laboratorium, Obliczenia naukowe

Autor: Piotr Klepczyk

Analiza problemu

Jednostka badawcza dużej firmy działającej w branży chemicznej prowadzi intensywne badania. Wynikiem tych badań są modele pewnych zjawisk chemii kwantowej. Rozwiązanie tych modeli, w pewnym szczególnym przypadku, sprowadza się do rozwiązania układu równań liniowych

$$\boldsymbol{Ax = b.}$$

dla danej macierzy współczynników $\boldsymbol{A} \in R^{n \times n}$ i wektora prawych stron $\boldsymbol{b} \in R^n$. Macierz $\boldsymbol{A}$ jest rzadką, tj. mającą dużą elementów zerowych, i blokową o następującej strukturze:

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

v = n/l, zakładając, że n jest podzielne przez l, gdzie l jest rozmiarem wszystkich kwadratowych macierzy wewnętrznych (bloków): $\boldsymbol{A_k}$, $\boldsymbol{B_k}$ i $\boldsymbol{C_k}$. Mianowicie, $\boldsymbol{A_k} \in R^{l \times l}$, $k = 1, \ldots, v$ jest macierzą gęstą, 0 jest kwadratową macierzą zerową stopnia l, macierz $\boldsymbol{B_k} \in R^{l \times l}$, $k = 2, \ldots, v$ jest następującej postaci:

$$\boldsymbol{B}_k = \begin{pmatrix} 0 & \cdots & 0 & b_1^k \\ 0 & \cdots & 0 & b_2^k \\ \vdots & & \vdots & \vdots \\ 0 & \cdots & 0 & b_\ell^k \end{pmatrix},$$

$\boldsymbol{B_k}$ ma tylko jedną, ostatnią, kolumnę niezerową. Natomiast $\boldsymbol{C_k} \in R^{l \times l}$, $k = 1, \ldots, v - 1$ jest macierzą diagonalną:

$$\boldsymbol{C}_k = \begin{pmatrix} c_1^k & 0 & 0 & \cdots & 0 \\ 0 & c_2^k & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c_{\ell-1}^k & 0 \\ 0 & \cdots & 0 & 0 & c_\ell^k \end{pmatrix}.$$

Firma napotkała problem z efektywnym rozwiązaniem problemu $\boldsymbol{Ax = b}$, gdzie $\boldsymbol{A}$ jest postaci (1). Chodzi przede wszystkim o wymagania czasowe i pamięciowe, które wynikają z bardzo dużego rozmiaru macierzy (n jest bardzo duże). W konsekwencji wyklucza to pamiętanie macierzy $\boldsymbol{A}$ jako tablicę $n \times n$ oraz użycie standardowych (bibliotecznych) algorytmów dla macierzy gęstych (tj. takich, gdzie nie zakłada się dużej liczby elementów zerowych). Zatem jedynym podejściem do rozwiązania tego problemu jest specjalna struktura pamiętająca efektywnie macierz $\boldsymbol{A}$, która pamięta tylko elementy niezerowe i adaptacja standardowych algorytmów tak, aby uwzględniały postać macierzy (1), tj. jej rzadkość, regularność występowania elementów zerowych i niezerowych (jeżeli ` jest stałą, to czas $O(n^3)$ można zredukować do O(n)).

1. Zadanie 1

1.1. Cel

Celem zadania jest napisanie funkcji rozwiązująca układ $\boldsymbol{Ax} = \boldsymbol{b}$ metodą eliminacji Gaussa z uwzględnieniem postaci macierzy. Funkcja ma działać z częściowym wyborem elementu głównego i bez. Program ma wczytywać macierz i wektor z pliku, a wynik zapisać do pliku. Funkcja ma mieć możliwość wyznaczenia wektora b jeśli ten nie został podany.

1.2. Realizacja

Macierze i wektory przechowywane są jak *SparseMatrixCSC* i *SparseVector*, typy te są oferowane w języku Julia i przechowują one wartości w sposób efektywny, to znaczy przechowują wielkość macierzy/wektora i tylko te dane których wartość jest różna od domyślnej, w tym wypadku zero oraz indeksy tych wartości.

Do realizacji zadania wykorzystano algorytm eliminacji Gaussa przedstawiony na wykładzie, ze przedziałami dobranymi dal specyficznej postaci analizowanej macierzy. Sam algorytm eliminacji Gaussa polega na uzyskaniu macierzy górno trójkątnej, ponieważ przy analizie macierzy jako reprezentacji układu równań w łatwy sposób można odczytać rozwiązanie. Taką macierz uzyskujemy przez wielokrotne odejmowanie aktualnego wiersza od wierszy poniżej, dla pozycji $a_{i,i}$ współczynnik przez który pomnożymy macierz przy odejmowaniu dla wiersza j-tego wyznaczamy wzorem:

$$m = \frac{a_{j,i}}{a_{i,i}}$$

Taką samą operację wykonujemy również na wektorze $\boldsymbol{b}$, który traktujemy jako dodatkową kolumnę macierzy. W celu lepszego zobrazowania działania algorytmu dla jednego kroku, poniżej pokazano schemat na przykładzie ogólnym:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_1 \\ a_{2,1}-\frac{a_{2,1}}{a_{1,1}}a_{1,1} & a_{2,2}-\frac{a_{2,1}}{a_{1,1}}a_{1,2} & \dots & a_{2,n}-\frac{a_{2,1}}{a_{1,1}}a_{1,n} & b_2-\frac{a_{2,1}}{a_{1,1}}b_1 \\ \dots & \dots & \dots & \dots & \dots \\ a_{n,1}-\frac{a_{n,1}}{a_{1,1}}a_{1,1} & a_{n,2}-\frac{a_{n,1}}{a_{1,1}}a_{1,2} & \dots & a_{n,n}-\frac{a_{n,1}}{a_{1,1}}a_{1,n} & b_n-\frac{a_{n,1}}{a_{1,1}}b_1 \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_1 \\ 0 & a_{2,2}-\frac{a_{2,1}}{a_{1,1}}a_{1,2} & \dots & a_{2,n}-\frac{a_{2,1}}{a_{1,1}}a_{1,n} & b_2-\frac{a_{2,1}}{a_{1,1}}b_1 \\ 0 & \dots & \dots & \dots & \dots \\ 0 & a_{n,2}-\frac{a_{n,1}}{a_{1,1}}a_{1,2} & \dots & a_{n,n}-\frac{a_{n,1}}{a_{1,1}}a_{1,n} & b_n-\frac{a_{n,1}}{a_{1,1}}b_1 \end{bmatrix}$$

Jak widać efektem algorytmu jest wyzerowanie kolumny poniżej elementu diagonali należącego do aktualnie analizowanego wiersza. W algorytmie stworzonym do eliminacji Gaussa dla zadanych specyficznym macierzy nałożono ograniczenia dla zakresu w którym algorytm przeprowadza obliczenia. Warto zauważyć że poniżej diagonali w każdym wierszu jest maksymalnie *l* wierszy, gdzie *l* odpowiada wielkości macierzy wewnętrznej tak jak zostało to zaprezentowane w analizie problemu. Pozostałe wartości w kolumnie to 0, więc nie ma potrzeby stosowaniu wzory w tym miejscu, tak otrzymujemy pierwsze ograniczenie, algorytm działa więc maksymalnie na *l* wierszach, warto zauważyć jednak że macierz układa się w „schodki ” o wysokości *l*. Ze względu na ten fakt zakres działania algorytmu (zmienna *MaxRow*) na wierszach dla i-tego wiersza określa równanie:

$$MaxRow = l + \left\lfloor \frac{i}{l} \right\rfloor * l$$

Kolejnym ograniczeniem na tym etapie na obszar działania algorytmu jest ilość analizowanych kolumn w wierszu, analizując konstrukcję macierzy łatwo zauważyć że największa możliwa odległość między elementem diagonali, a ostatnim niezerowym elementem tego wiersza to *l*. Stąd równanie określające maksymalny zakres kolumn (zmienna MaxColumn) dla algorytmu w i-tym wierszu:

$$MaxColumn = l + i$$

Dzięki temu algorytm działa tylko na obszarach gdzie są uzyskiwane niezerowe wyniki.

Mając już macierz górno trójkątną możemy przejść do obliczania wartości wektora *x*. Algorytm wyliczający wartości w wektorze działa od dołu macierzy, ponieważ w ostatnim wierszu mamy tylko jeden element, dla macierzy o wielkości *n x n* znajduje się on na pozycji *n,n*, wartość ta odpowiada równaniu:

$$a_{n,n} = x_n$$

W kolejnym kroku rozwiązujemy równanie z wiersza wyżej gdzie są dwie niewiadome, ale jedną z nich mamy obliczoną z kroku poprzedniego. Tak rozwiązując układy równań otrzymamy wszystkie wartości wektora *x*. Tutaj również zastosowano ograniczenie na liczbie analizowanych kolumn macierzy, takie samo jak poprzednio, gdyż tylko w tym obszarze występują elementy niezerowe.

Kolejną funkcjonalnością jaką posiada program jest obliczanie wektora x przy pomocy eliminacji Gaussa z częściowym wyborem elementu głównego. Sam algorytm jest prawie identyczny do tego opisanego powyżej, z tym że dla analizowanego elementu diagonali najpierw sprawdzamy elementy w kolumnie poniżej i wybieramy ten dla którego wartość bezwzględna jest największa i zamieniamy jego pozycję z tym w którym jesteśmy, zmieniamy też odpowiednie wartości w wektorze *b*. W algorytmie nie występuje jednak fizyczne przepisanie wartości w macierzy, zastosowano podmianę indeksów przy użyciu tablicy, którą nazwano *swaped*, jest ona długości n i na początku pod danym indeksem znajduje się wartość odpowiadająca wartości indeksu, czyli *swaped*[i] = i. Dzięki temu oszczędzamy czas i pamięć która została by zużyta na znalezienie i zamianę miejsca każdego elementu wiersza. Tak więc zamiana wierszy to tak naprawdę zamiana wartości między indeksami tablicy, gdzie indeksy odpowiadają numerowi wiersza. Potem przy odczytywaniu wartości w miejsce indeksu wiersza podajemy wartość występującą w tablicy pod tym indeksem. Tutaj również zastosowano ograniczenia na wiersze i kolumny. Ograniczenie na wierszach macierzy analizowanych dla aktualnego jest takie samo jak poprzednio. Natomiast na kolumny ograniczenie musiało być zmienione ze względu na swapowanie wierszy, ograniczenie zostało stworzone w oparciu o to jakie wiersze mogą być ze sobą zamieniane, tak więc ograniczenie na liczbę kolumn musi być „schodkowe” po wielkości *l*. Jest tak ponieważ ostatni wiersz w schodku ma ostatni element o *l-1* pozycji dalej niż pierwszy, a w wyniku pracy algorytmu może on „wylądować” na każdej pozycji. Tak więc ograniczenie na kolumnach określane jest równaniem:

$$MaxColumn = 2l + \left\lfloor \frac{i-1}{l} \right\rfloor * l$$

Wartości wektora x są wyliczane tak jak bez częściowego wyboru elementu głównego, z tym że używana jest tablica swapów dla indeksów wektorów i wierszy macierzy.

Program wczytuje macierz z pliku jak to było zadane w poleceniu do zmiennej, która jest typu SparseMatrixCSC, który został opisany wcześniej. To samo tyczy się wektora, z tym że program posiada funkcję wyznaczenia wektora *b*, gdy nie został on podany. Jest on wyznaczany poprzez rozwiązanie układu równań $\boldsymbol{Ax = b}$, gdzie $\boldsymbol{x}$ jest wektorem samych jedynek. Tutaj również zastosowano ograniczenie na kolumnach, takie samo jak w przypadku przeprowadzania eliminacji Gaussa bez wyboru elementu głównego.

Algorytm testowano na danych ze strony kursu. W testach mierzono czas działania algorytmów oraz średni błąd bezwzględny uzyskanych wyników. Poniżej przedstawiono tabele prezentujące wyniki testów.

n	Czas	Pamięć	Błąd bezwzględny
Bez wyboru elementu głównego			
16	0.026806 s	43.047 KiB	2.671474153004283e-15
10000	0.174674 s	2.501 MiB	3.9918734984212275e-15
50000	4.685829 s	2.002 MiB	2.6855939694314656e-15
Z wyborem elementu głównego			
16	0.044721 s	44.891 KiB	4.0939474033052647e-16
10000	0.203442 s	2.578 MiB	3.011813021203125e-16
50000	4.699558 s	2.383 MiB	3.1890046159332995e-16

Tab1.2.1 Tablica wyników testów

Jak widać czas działania algorytmu rośnie wraz z wielkością macierzy, co jest zrozumiałe gdyż algorytm musi wykonać więcej razy operacje i mimo zastosowanych typów do przechowywania typów to ilość danych do zapamiętania rośnie wraz z wielkością macierzy. Można też zauważyć że wybór elementu głównego dla każdego przypadku daje o rząd dokładniejszy wynik, ale kosztem nieco dłuższego czasu i większego zużycia pamięci, co też jest zrozumiałe ze względu na swapy oraz sposób ich pamiętania. Można jednak stwierdzić że program działa poprawnie, a w obu przypadkach błąd jest mały. Błąd ma miejsce ze względu na występowanie dzielenia i mnożenia gdzie może dojść do zaokrąglenia związanego z arytmetyką w której przeprowadzane są obliczenia.

2. Zadanie 2

2.1. Cel

Celem zadania jest napisanie funkcję wyznaczającą rozkład LU macierzy metodą eliminacji Gaussa, uwzględniają specyficzną postać zadanej macierzy. Funkcja ma działać z wyborem elementu głównego, jak i bez.

2.2. Realizacja

Do realizacji zadania jako bazę wykorzystano program z Zadania 1, ponieważ przeprowadzając zwykłą eliminację Gaussa otrzymujemy macierz górno trójkątną odpowiadającą macierzy *U* z rozkładu *LU*. Dla efektywnego przechowywania macierze *L* i *U* są przechowywane jako jedna macierz z wykorzystaniem typu *SparseMatrixCSC*. Zmiany jakie należało wdrożyć w algorytmie z Zadania 1 by przeprowadzał rozkład *LU* to usunięcie wszystkich operacji na

wektorze *b* oraz zamiast zerowanie kolumny poniżej aktualnego elementu diagonali to wstawiamy tam wynik wyrażenia (ale $a_{i,i}$):

$$m = \frac{a_{j,i}}{a_{i,i}}$$

Dzięki temu poniżej diagonali naszej macierzy otrzymujemy wartości odpowiadające wartością na macierzy *L* z rozkładu *LU*. Główna oś macierzy L powinna mieć wszystkie wartości równe 1, w tym wypadku wartości te są domyślne. Dla efektywnego działania algorytmu ze względu na specyficzną postać macierzy w oby przypadkach, to jest dla wyboru elementu głównego jak i bez wyboru, zastosowano te same ograniczenia na wiersze i kolumny co wcześniej. Dzięki temu algorytm działa tylko tam gdzie jest możliwość uzyskania wyniku niezerowego. Testy tego algorytmu sprawdzają poprawność równania *A = L*U*, by można było przeprowadzić te obliczenia została stworzona specjalna funkcja która rozdziela macierz *LU* na dwie osobne *L* i *U*. Co ułatwi nam wymnożenie ich ze sobą, a następnie porównanie uzyskanego wyniku z macierzą *A*. Program porównuje wszystkie wartości pod tymi samymi indeksami, lecz nie szuka on dokładnej równości tylko przybliżonej przy pomocy symbolu „≈”, zastosowano go ponieważ mogą wystąpić niewielkie odchylenia związane z zaokrągleniem w arytmetyce. Testy zwracają czas wykonania algorytmu rozkładu LU oraz jego poprawność.

n	Czas	Czy poprawny?
Bez wyboru elementu głównego		
16	0.018870 s	tak
10000	0.167198 s	tak
50000	4.747816 s	tak
Z wyborem elementu głównego		
16	0.028225 s	tak
10000	0.174419 s	tak
50000	4.630965 s	tak

Tab2.2 Tablica wyników testów

Wyniki testów są pozytywne co oznacza że rozkład *LU* jest poprawny.

3. Zadanie 3

3.1. Cel

Celem zadania jest napisanie funkcję rozwiązującą układ równań *Ax = b*, gdy został wyznaczony rozkład *LU*, uwzględniając specyficzną postać macierzy.

3.2. Realizacja

Do realizacji zadania wykorzystano funkcje do wyznaczania rozkładu *LU* z zadania poprzedniego. Przy przeprowadzaniu rozkładu *LU* należy zmodyfikować dane na wektorze *b*, by można było przejść do rozwiązywania równania. Wektor b jest modyfikowany w sposób następujący (dla pozycji $a_{j,i}$ w macierzy po rozkładzie *LU*):

$$b_j = b_j - (a_{j,i} * b_i)$$

Działanie to jest wykonywane na tych samych ograniczeniach na wiersze co w Zadaniu 1 dla rozwiązywania zadania bez wyboru elementu głównego. Powyższe równanie jest tak naprawdę równaniem które było stosowane dla wektora i macierzy w Zadaniu 1, wystarczy zauważyć tylko że po przeprowadzeniu rozkładu *LU* w miejscu $a_{j,i}$ znajduje się wartość:

$$a_{j,i} = \frac{a_{j,i}}{a_{i,i}}$$

Po zastosowaniu powyższej operacji na wektorze *b*, można przejść do obliczania wartości wektora *x*. Wartość ta jest obliczana dokładnie w taki sam sposób jak w Zadaniu 1, z tymi samymi ograniczeniami na kolumny. Analogiczne operacje są wykonywane dla przypadku z częściowym wyborem elementu głównego, z tym że stosowano tablice swapów w miejscach indeksów wektora b oraz w miejscu indeksu wiersza macierzy. Zastosowano również ograniczenia na kolumny takie jak w Zadaniu 1 dla części z wyborem elementu głównego. Program był testowany dwuetapowo, to znaczy najpierw była sprawdzana poprawność rozkładu *LU*, gdy rozkład był poprawny testowano wynik uzyskany przez algorytm wyliczający wektor *x*. Testy poprawności rozkładu opisano w Zadaniu 2, tutaj zastosowano dokładnie te same dane testowe. Wyniki testów wliczania wektora *x*:

n	Czas	Pamięć	Błąd bezwzględny
Bez wyboru elementu głównego			
16	0.014817 s	27.109 KiB	2.671474153004283e-15
10000	0.006139 s	513.438 KiB	3.9918734984212275e-15
50000	0.033057 s	2.002 MiB	2.6855939694314656e-15
Z wyborem elementu głównego			
16	0.022741 s	27.938 KiB	4.0939474033052647e-16
10000	0.008838 s	513.438 KiB	3.011813021203125e-16
50000	0.046122 s	2.002 MiB	3.1890046159332995e-16

Tab3.2.1 Tablica wyników testów

Jak widać uzyskane wyniki są poprawne, to znaczy błędy są niewielkie, a błąd bezwzględny w przypadku wyboru elementu głównego jest o rząd mniejszy. Niewielkie czasy wykonania mają tutaj miejsce gdyż nie ma operacji modyfikacji danych na macierzy, a tylko odczyt potrzebnych danych do obliczeń, czasy też są tak niskie ze względu na zastosowane ograniczenia na analizowane obszary macierzy co poprawia wydajność czasową algorytmu.