Obliczenia Naukowe

Laboratorium 5

Bartosz Grelewski

Styczeń 2023

1 Zadanie 1

1.1 Opis 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.

Ax=b

Dla danej macierzy współczynników $A \in R^{n \times n}$ i wektora prawych stron $b \in R^n, n \geq 4$. Macierz A jest rzadką, mającą dużą elementów zerowych, i blokową o następującej strukturze:

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

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

$$\boldsymbol{B}_k = \begin{pmatrix} b^k_{1\,1} & \cdots & b^k_{1\,,\ell-2} & b^k_{1\,\ell-1} & b^k_{1\,\ell} \\ 0 & \cdots & 0 & 0 & b^k_{2\,\ell} \\ \vdots & & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & b^k_{\ell\,\ell} \end{pmatrix},$$

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

$$\boldsymbol{C}_k = \begin{pmatrix} c^k_1 & 0 & 0 & \cdots & 0 \\ 0 & c^k_2 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & c^k_{\ell-1} & 0 \\ 0 & \cdots & 0 & 0 & c^k_\ell \end{pmatrix}.$$

Firma napotkała problem z efektywnym rozwiązaniem problemu $Ax = b$, gdzie 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 A jako tablicę n × 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 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 l jest stałą, to czas $O(n^3)$ można zredukować do $O(n)$)

1. Napisać funkcję rozwiązującą układ Ax = b metodą eliminacji Gaussa uwzględniającą specyficzną postać (1) macierzy A dla dwóch wariantów:
(a) bez wyboru elementu głównego,
(b) z częściowym wyborem elementu głównego.

1.2 Wstęp

Z treści zadania wynika potrzeba zaimplementowania dwóch głównych algorytmów, które pozwalają nam na optymalniejsze obliczenie układu Ax=b. Pierwszy (a) jest podstawą dla drugiego, który jest jego lepszą wersją ze względu na zawarcie chociażby tego, że na przekątnej mogą pojawiać się zera i nie następuje próba dzielenia przez nie. Poniżej znajdują się bardziej szczegółowe wyjaśnienia metod do implementacji w formie podstawowej, a następnie z modyfikacjami, które wynikają z treści zadania i zachowania odpowiednich złożoności obliczeniowych. Nie umieszczałem w sprawozdaniu kodów źródłowych.

1.3 Użyte struktury wobec macierzy rzadkich

Sparse Arrays
Aby efektywnie przechowywać macierz rzadką korzystałem z biblioteki podstawowej w języku Julia (Sparse Arrays). Zaletą tej biblioteki jest to, że pamięta ona tylko niezerowe elementy naszej macierzy. Zakładamy wtedy, że dostęp do elementu macierzy mamy wtedy w czasie stałym. Sposób implementacji tej struktury pozwala na szybszy dostęp do elementów macierzy poprzez odwoływanie się do nich przez kolumny niż przez wiersze. Dla celów badania złożoności czasowej naszych algorytmów przyjęliśmy, że dostęp do elementu takiej macierzy jest równy $O(1)$ Chociaż doskonale wiemy, że niekoniecznie tak jest.

1.4 Operacje wejścia i wyjścia

Wczytywanie danych
Używam metody *open*, która znajduje się w bibliotece domyślnej *Julii*. Wczytuje linijka po linijce z uwzględnieniem delimiterów pod postacią spacji. Zapisuje dane do zmiennej *A*, która jest typu Sparse Array.

Zapis danych Podczas zapisu danych używam analogicznie metody z biblioteki podstawowej *open* jedynie z dopiskiem "w" (write). W tym miejscu jest też metoda odpowiedzialna za zapis błędu do pliku jeżeli b (wektor prawych stron) jest generowany.

1.5 Wytłumaczenie na czym polega podstawowa metoda Gaussa

Ta metoda opiera się w głównej mierze na odejmowaniu kolejnych wierszy, które są przemnożone przez czynnik od tych znajdujących się pod nimi, tak aby doprowadzić do wyzerowania wszystkich wartości pod przekątną.
Czyli zwykły gauss polega na doprowadzeniu macierzy z układu równań do postaci w trójkącie. Zeruje się po kolei wszystkie współczynniki poniżej przekątnej. Zajmuje to $O(n^3)$ czasu, gdzie n to rozmiar macierzy. Znając postać macierzy (macierz podana w treści zadania) nie trzeba zerować wszystkich współczynników, ponieważ są już one zerowe. Wystarczy wyzerować l współczynników pod przekątną. Dzięki temu jesteśmy w stanie osiągnąć złożoność $O(n)$, ponieważ w testach zakładamy, że l jest stałą.

Jest to wiedza z wykładu, dlatego też posłużę się przykładem jak wygląda Metoda Eliminacji Gaussa bez wyboru elementu głównego. Pierwszy etap eliminacji polega na usunięciu niezerowych elementów.

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{A} \in \mathbb{R}^{n\times n}, \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{b} \in \mathbb{R}^n, \det(\boldsymbol{A}) \neq 0$$

$$\boldsymbol{A}^{(1)} = \boldsymbol{A},\ \boldsymbol{b}^{(1)} = \boldsymbol{b}$$

$$\boldsymbol{A}^{(1)}\boldsymbol{x} = \boldsymbol{b}^{(1)} \quad \begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\ a_{21}^{(1)}x_1 & + & a_{22}^{(1)}x_2 & + & \cdots & + & a_{2n}^{(1)}x_n & = & b_2^{(1)} \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}^{(1)}x_1 & + & a_{n2}^{(1)}x_2 & + & \cdots & + & a_{nn}^{(1)}x_n & = & b_n^{(1)} \end{array}$$

Eliminujemy zmienną x_1 z równań od 2-go do n-tego. Mnożymy 1-sze równanie przez $l_{i1} = a_{i1}^{(1)}/a_{11}^{(1)}$ gdzie $i = 2, ..., n$ i odejmujemy od pozostałych.

W drugim kroku otrzymujemy:

$$\boldsymbol{A}^{(2)}\boldsymbol{x} = \boldsymbol{b}^{(2)} \quad \begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ & & \vdots & & & & \vdots & & \vdots \\ & & a_{n2}^{(2)}x_2 & + & \cdots & + & a_{nn}^{(2)}x_n & = & b_n^{(2)} \end{array}$$

Analogicznie eliminujemy zmienną x_2 z równań od 3-go do n-tego. Mnożymy 2-gie równanie przez $l_{i2} = a_{i2}^{(2)}/a_{22}^{(2)}$ gdzie $i = 3, ..., n$

W uogólnieniu dostajemy po k-1 krokach:

$$\boldsymbol{A}^{(k)}\boldsymbol{x} = \boldsymbol{b}^{(k)} \quad \begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ & & & \ddots & & & \vdots & & \vdots \\ & & & & a_{kk}^{(k)}x_k & +\cdots+ & a_{kn}^{(k)}x_n & = & b_k^{(k)} \\ & & & & \vdots & & \vdots & & \vdots \\ & & & & a_{nk}^{(k)}x_k & +\cdots+ & a_{nn}^{(k)}x_n & = & b_n^{(k)} \end{array}$$

Eliminujemy zmienną x_k z równań od k+1-tego do n-tego. Mnożymy k-te równanie przez:

$l_{ik} = a_{i2}^{(k)}/a_{kk}^{(k)}$ gdzie $i = k+1, ..., n$ i odejmujemy od pozostałych.

Po n-1 krokach dostajemy układ z macierzą górno-trójkątną:

$$\boldsymbol{A}^{(n)}\boldsymbol{x} = \boldsymbol{b}^{(n)}$$

$$\begin{array}{ccccccccc} a_{11}^{(1)}x_1 & + & a_{12}^{(1)}x_2 & + & \cdots & + & a_{1n}^{(1)}x_n & = & b_1^{(1)} \\ & & a_{22}^{(2)}x_2 & + & \cdots & + & a_{2n}^{(2)}x_n & = & b_2^{(2)} \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & a_{nn}^{(n)}x_n & = & b_n^{(n)} \end{array}$$

Następnym etapem jest rozwiązanie tego co nam wyszło. Dlatego iterujemy od n do 1 i korzystamy z: $x_n = b_n/a_{nn}$ oraz $x_k = (b_k - \sum_{j=k+1}^{n} a_{kj}x_j)/a_{kk}$

1.6 Metoda Gaussa zoptymalizowana pod podany problem

Standardowa wersja procesu eliminacji ma złożoność $O(n^3)$ ze względu na to, że k-ty wiersz musimy odjąć od n-k wierszy poniżej niego, a każde takie odejmowanie to jest nadpisywanie każdej z n+1 wartości w danym wierszu.

Po zrozumieniu jak działa Metoda Gaussa należało przejść do przepisania jej pod treść zadania. Uwzględniając macierze w postaci rzadkiej na początku zostanie opisana metoda bez wyboru elementu głównego.

1.7 Metoda Gaussa bez wyboru elementu głównego

Powołując się na treść zadania wiemy, że macierz A to tak naprawdę 3 rodzaje podmacierzy. A_k, B_k, C_k są podmacierzami kwadratowymi o rozmiarze $l \geq 2$. A są w całości niezerowe (jest to l^2 el. niezerowych), B mają niezerowe ostatnie kolumny i pierwszy niezerowy wiersz (2*l el. niezerowych), C mają niezerową przekątną (l elementów niezerowych). Reszta komórek to zera. Chcemy uzyskać macierz trójkątną. Jest to specyficzna macierz trójkątna bo taka, która zawiera zera pod przekątną więc w algorytmie żeby nie zerować wszystkiego to możemy zrobić to na zasadzie usuwania w pionie 4,3,2,4,4,3,2,4... elementów. Czyli mając pierwszą pętle "for" w kodzie musimy pamiętać, że wykona się ona dokładnie n-1 razy oraz odpowiednio usuwamy po 4 elementy. Schodząc do kolejnej zagnieżdżonej pętli jej wykonanie to l razy, później na ostatniej pętli (3-ciej) jest analogiczna sytuacja, co do drugiej, jednak jest to jej min, czyli może się wykonać nawet mniej razy ale nie przekroczy l. Jest to zabieg optymalizacyjny, który polega na ominięciu części iloczynów przez elementy zerowe. Chodzi oczywiście o te, które znajdują się po prawej stronie od wartości, na której zatrzymała się w danym momencie iteracja 2-giej pętli. Czyli l jest stałe. Co za tym idzie wyznaczenie macierzy wykonuje się w czasie liniowym. Newralgicznym momentem w algorytmie jest warunek $min(k+1, n)$, który

odpowiada za to żeby nie wyjść poza rozmiar macierzy.
Poniżej znajduje się macierz A, l = 4.

$$\begin{pmatrix} a_{11}^1 & a_{12}^1 & a_{13}^1 & a_{14}^1 & c_{11}^1 & 0 & 0 & 0 & \dots \\ a_{21}^1 & a_{22}^1 & a_{23}^1 & a_{24}^1 & 0 & c_{22}^1 & 0 & 0 & \dots \\ a_{31}^1 & a_{32}^1 & a_{33}^1 & a_{34}^1 & 0 & 0 & c_{33}^1 & 0 & \dots \\ a_{41}^1 & a_{42}^1 & a_{43}^1 & a_{44}^1 & 0 & 0 & 0 & c_{44}^1 & \dots \\ b_{11}^1 & b_{12}^1 & b_{13}^1 & b_{14}^1 & a_{11}^2 & a_{12}^2 & a_{13}^2 & a_{14}^2 & \dots \\ 0 & 0 & 0 & b_{24}^1 & a_{21}^2 & a_{22}^2 & a_{23}^2 & a_{24}^2 & \dots \\ 0 & 0 & 0 & b_{34}^1 & a_{31}^2 & a_{32}^2 & a_{33}^2 & a_{34}^2 & \dots \\ 0 & 0 & 0 & b_{44}^1 & a_{41}^2 & a_{42}^2 & a_{43}^2 & a_{44}^2 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

W moim rozwiązaniu znajdują się dwa etapy podzielone odpowiednio na metody: **gauss** i **solve_gauss**. Każda z nich ma swoją funkcję. Jednak etapowo wygląda to tak, że zaczynamy od **gauss** i potem rozwiązujemy w drugim etapie, który jest naszym **solve_gauss**.

Złożoność liczymy tak: (gauss) $O(n \cdot l^2)$ + (solve_gauss) $O(n \cdot l) = O(n \cdot l^2) = O(n)$
Zaczęliśmy od $O(n^3)$, później po skorzystaniu z treści zadania możemy przyjąć l za stałą więc schodzimy ze złożonością do $O(n \cdot l^2)$ co uznajemy za $O(n)$.

Solve_gauss - opis
Drugą fazą algorytmu jest rozwiązanie przekształconego układu. Wyznaczamy x_n z ostatniego równania
$x_n = \frac{b_n}{a_{nn}}$. Dalej wyznaczamy x_k dla $k = n-1, ..., 1$

$$x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj} x_j}{a_{kk}}$$

1.8 Gauss z częściowym wyborem elementu głównego

Ten algorytm jest rozszerzeniem bazowej metody eliminacji Gaussa o odpowiednie spermutowanie kolejności wierszy macierzy tak aby na przekątnej znajdowały się niezerowe elementy oraz żeby były odpowiednio duże (żeby nie weszły z zakres zera maszynowego).

Reasumując, można poradzić sobie z ewentualnymi 0 występującymi na diagonali macierzy. Można zauważyć, że warunek $a_{kk} \neq 0$ na ogół nie zapewnia numerycznej stabilności algorytmu. W celu zapobiegania takim sytuacjom stosuje się klasyczny algorytm Gaussa rozszerzony o wybór elementu głównego w kolumnie. Polega on na znalezieniu w kolumnie największego (z dokładnością do wartości bezwzględnej) elementu i odpowiednim przestawieniu wierszy macierzy tak, aby wybrany element znalazł się w określonym miejscu na diagonali. Możemy wyrazić to następującym wzorem: $|a_{kk}| = |a_{s(k),k}| = max(|a_{ik}| : i = k, ..., n)$ gdzie s(k) jest wektorem permutacji, w którym pamiętana jest kolejność przestawień.

1.9 Gauss z częściowym wyborem elementu głównego - zoptymalizowany

Algorytm eliminacji Gaussa z częściowym wyborem elementu głównego jest bardzo podobny do przedstawionego powyżej algorytmu z tą małą różnicą, że dochodzi tutaj jeszcze wektor permutacji, w którym pamiętane jest przestawienie elementów w kolumnach. Modyfikacja ma

na celu zapobieganie występowaniu zer na przekątnej macierzy.

Opis parametrów:
A - macierz rzadka A,
b - wektor prawych stron
n - rozmiar macierzy A
l - rozmiar podmacierzy

Wektor rozwiązań układu x - jest wynikiem zwracanym przez metodę.

W metodzie na początku uzupełniamy wektor perm[i] elementami od 1 do n. Następnie wchodzimy w pierwszą pętle, która wykonuje się od 1 do n-1 (przechodzenie po wszystkich kolumnach). Następuje szukanie największego elementu w kolumnie wraz z wierszem, w którym się znajduje. Później następuje zamiana wierszy w macierzy A i wektorze b. Po swapie znajdują się kolejne dwie pętle, które są algorytmem eliminacji Gaussa z poprzedniego punktu. Złożoność algorytmu będzie podobna jak wcześniej, ale z racji na obecność dodatkowej pętli, w której wybierany jest element główny, oraz faktu, że pętla wykona się 2l razy (ze względu na to, że ostatni, niezerowy element, można stworzyć w kolumnie o indeksie 2l, gdy eliminujemy czynniki niezerowe z pierwszych l - 1 kolumn), wyniesie ona nie więcej niż O(n), aczkolwiek będzie nieco większa (z dokładnością do stałej), ponieważ wykonujemy nieco wiecej operacji.

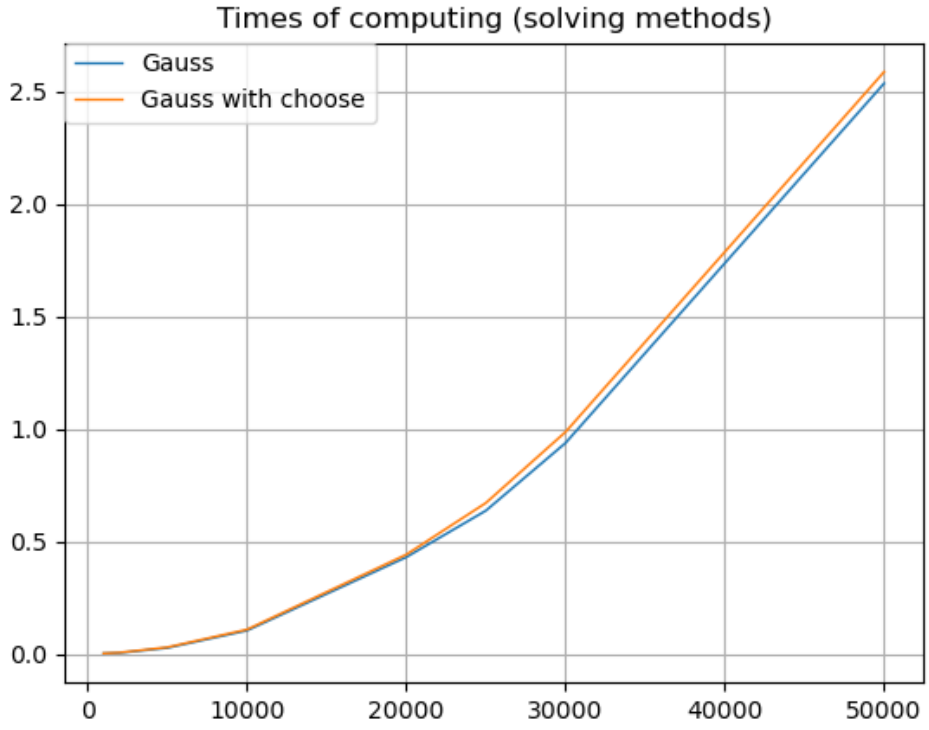
W algorytmie Gaussa z częściowym wyborem elementu głównego dochodzi nam jeden FOR na drugim poziomie zagnieżdżenia, która wykonuje się zawsze l+1 razy. Dodatkowo we wszystkich późniejszych pętlach FOR na poziomie drugim liczba sprawdzanych indeksów wzrosła o l (z l do l + l). Wynika to z faktu, iż mogło dojść do zamiany wiersza k-tego z wierszem k+l-tym. W tej sytuacji dochodzi do zamian w całych wierszach, co ma znaczenie ze względu na strukturę macierzy C_k, która jest macierzą diagonalną. Weźmy pod uwagę sytuację, gdy $k = 1$, a maksymalną wartość znajduje się k=l. Wtedy należy zamienić ze sobą wiersze 1 i l, co prowadzi również do zamiany elementów znajdujących się w macierzy C_k w rozpatrywanym wierszu ostatni element będzie znajdował się na pozycji k+l-tej. Dlatego też mamy złożoność, która wynosi nie więcej niż O(n), aczkolwiek będzie nieco większa (z dokładnością do stałej w porównaniu z metodą Gaussa powyżej), ponieważ wykonujemy nieco wiecej operacji.

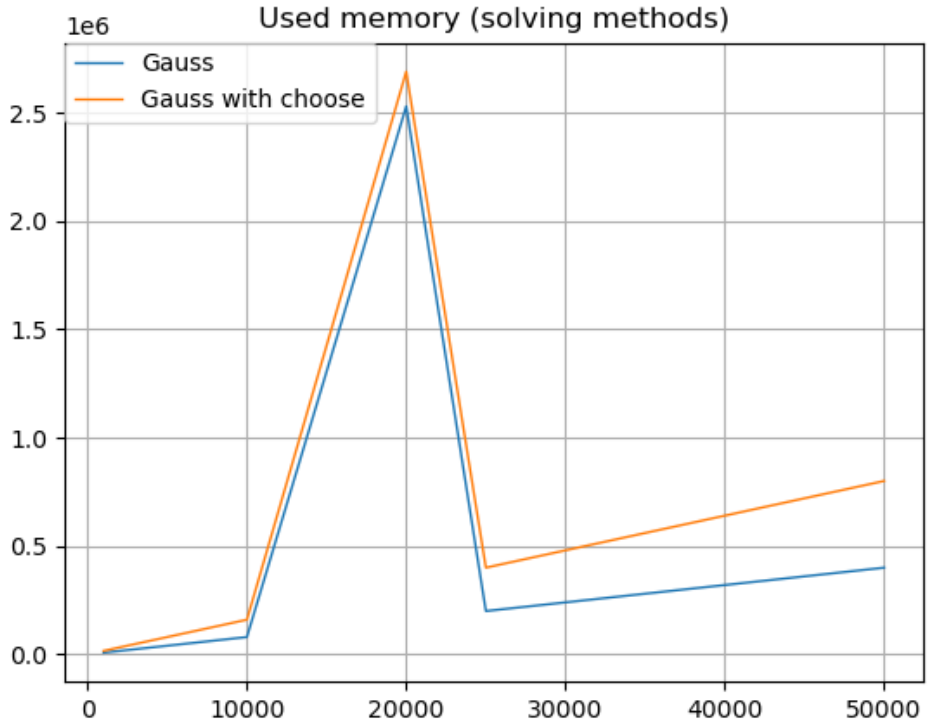
1.10 Porównanie algorytmów

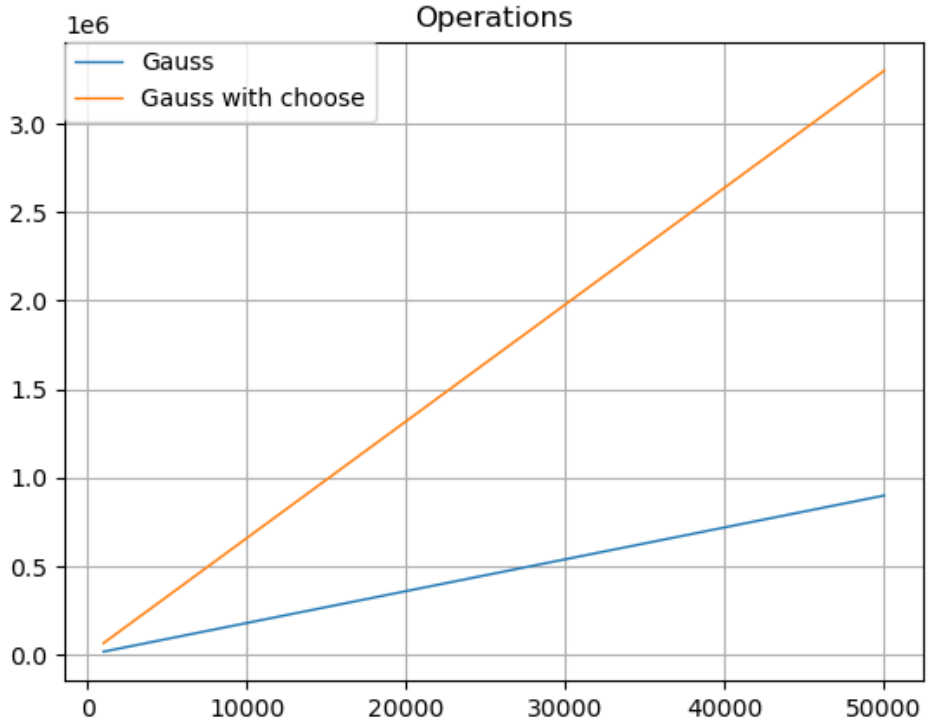
Aby dobrze porównać dwa algorytmy zastosowałem szereg testów, które zbadały w prosty sposób ich złożoność pamięciową/czasową. Użyłem *Test* z *Base* języka *Julia* pod postacią *@timed*, które samo w sobie jest miłym ułatwiającym pomiar makrem. Korzystając ze struktury Sparse Array założyłem, że czas dostępu jest liniowy. Testy przeprowadzone zostały przy generowaniu macierzy wielkości 1000, 2000, 5000, 10000, 20000, 25000, 30000, 50000 oraz generowanych osobno wektorów prawych stron.

Ponadto zrobiłem też testy jednostkowe. Korzystałem z funkcji *@testisapprox*, która sprawdzała zgodność wektora wynikowego funkcji $solve_gauss$ oraz wektora jedynek (jako poprawne rozwiązanie). Sprawdzana jest ich odległość względna/bezwględna od siebie, czy mieszczą się w ustalonej tolerancji.

1.11 Wykresy


Times of computing (solving methods)
Gauss
Gauss with choose
2.5
2.0
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
50000


Used memory (solving methods)
1e6
Gauss
Gauss with choose
2.5
2.0
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
50000


Operations
1e6
Gauss
Gauss with choose
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
50000

1.12 Analiza wykresów

Można zauważyć, że wraz ze wzrostem rozmiaru macierzy algorytmy są bardziej efektywne (zużywają mniej zasobów) pod względem alokacji pamięci potrzebnej do wykonania operacji. Widać również, że liczba iteracji dla wszystkich metod jest zależnością liniową od rozmiaru macierzy A. Rzeczywisty czas wykorzystany przez programy jest funkcją kwadratową.

1.13 Wnioski

Porównując otrzymane wyniki zauważamy, iż sposób z wykorzystaniem częściowego wyboru elementu głównego zwraca poprawniejsze wyniki, z mniejszym błędem względnym, kosztem nieznacznie większego zużycia pamięci. Czasowo funkcje realizowane są w bardzo podobnym tempie. Dodatkowo zauważamy, iż algorytmy z częściowym wyborem radzą sobie w sytuacji, gdy na przekątnej znajdują się elementy zerowe, czego niestety nie można powiedzieć o ich odpowiednikach bez wcześniejszych rotacji.

Widzimy, że na rzeczywisty czas wykonania wpłynęło odwoływanie się do elementów umieszczonych w strukturze SparseArrays, przez co na wykresie widać kwadratowy czas wykonania algorytmów. Biorąc jednak pod uwagę założenie, że dostęp do elementów macierzy jest stały otrzymujemy, że rzeczywiście wprowadzone modyfikacje sprawiły, że klasyczne algorytmy rozwiązywania równań liniowych dobrze sprawdzają się dla zadanej macierzy rzadkiej A i, że liczba operacji z każdego z modyfikowanych algorytmów wynosi O(n) (n może być przemnożone przez jakieś stałe, ale złożoność nadal pozostaje liniowa).