Barzowska Monika, 238143
Bienias Jan, 238201
3I TE1

20.11.2017

Algorytmy numeryczne

Zadanie 2: Operacje na macierzach

1. Wstęp

Celem zadania było przeprowadzenie operacji na macierzach przy użyciu:
- języka `C++` i biblioteki `Eigen3` wspierającej operacje na macierzach,
- implementacji niezbędnych funkcji w wybranym języku programowania (tu `C#`).

2. Materiały i metody

2.1. Dodawanie i mnożenie

Testy zostały przeprowadzone dla 3 zadanych wzorów: *A*X*, `A*`*B*C* oraz `(A+B+C)*X`, gdzie litery `A`, `B`, `C` oznaczają macierze, a `X` to wektor. Testy zostały przeprowadzone na 3 typach zmiennych: `double`, `float` i na własnym typie nazywanym *`fraction`*, który przechowuje liczby w postaci ułamka. Typ ten ma bezstratną precyzję.

Obliczenia generowane w `C#` dla typów `double`, `float` i *`fraction`* były porównywane z wynikami wygenerowanymi w `C++ Eigen` dla typu `double`, poprzez wyznaczenie średniej wartości dla końcowej macierzy i odjecie od siebie wyników.

Liczby które zostały wykorzystane do generowania macierzy oraz wektora, mieściły się w zakresie `[1.0, 20.0]`. Generowane i wyrażone zostały w postaci dziesiętnej, typie `double`.

2.2. Rozwiązywanie układów równań liniowych

Zadanie dotyczyło rozwiązywania układu równań liniowych macierzy A i wektora B.
W `C#` zaimplementowane zostały 3 zadane wersje metody eliminacji Gaussa:
- bez wyboru elementu podstawowego (gaussBase),
- z częściowym wyborem elementu podstawowego (gaussPartial),
- z pełnym wyborem elementu podstawowego (gaussFull).

W `C++ Eigen` wykorzystano wbudowane funkcje liczące rozwiązanie poprzez częściowy i pełny wybór elementu podstawowego.

Wynikiem przyjętym za poprawny i bazowy był wektor B, do którego porównywano wektor B_1 otrzymywany z „odwrócenia” operacji Gaussa.

```
gauss(A, B) = X
A * X = B1
```

metoda 1

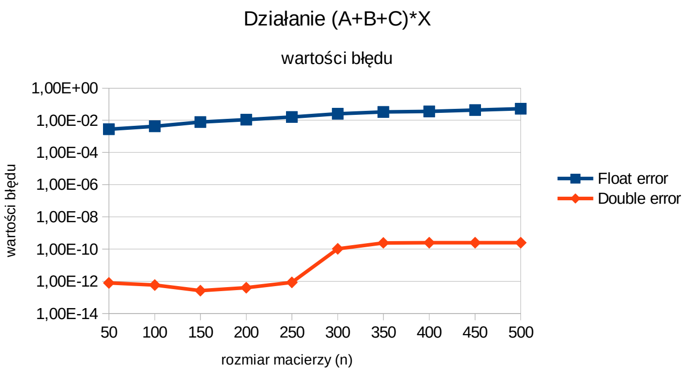
3. Wyniki i wnioski

Wyniki obliczeń zostały uśrednione z 5 próbek na każdy rozmiar macierzy. Taka ilość wystarczała, aby zauważyć tendencje zmian. Wyniki prezentowane dla typów `float` oraz `double` są obliczone dla wielkości macierzy $w \in [50, 500]$, zwiększane co 50. Wyniki prezentowane dla typu *`fraction`*, z uwagi na rosnący czas wykonywania operacji zależny od wielkości macierzy, są obliczone dla wielkości macierzy $w \in [10, 100]$, zwiększane co 10.

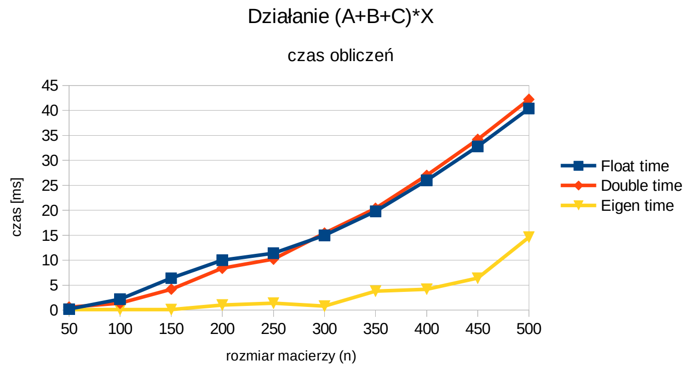
3.1. Dodawanie i mnożenie

3.1.1. Typy `double i float` – porównanie i czas

Wykres 1: Ukazuje wartości błędów powstałych podczas obliczania wyniku działania `(A+B+C)*X`.
Można zauważyć, że wraz ze wzrostem wielkości macierzy, błędy dla typów `double` i `float` delikatnie przybierają na wartości. W przypadku zarówno dodawania, jak i mnożenia wynika to z faktu obliczania kolejnych miejsc po przecinku, które nie mieszczą się już w zakresie danej liczby zmiennoprzecinkowej. Zarówno zmienna `double`, jak i `float` odrzucają końcowe cyfry mniejszej liczby aby pomieścić najistotniejsze wartości. Z uwagi na mniejszą precyzję typu `float` (7 miejsc) niż `double` (15-16 miejsc), w tym typie dużo bardziej zauważalne jest wspomniane odrzucanie, co za tym idzie – zaokrąglanie i w konsekwencji większe błędy.
Wykres 2: Wraz ze wzrostem rozmiaru macierzy, we wszystkich przypadkach (przedstawiony tylko jeden) odnotowano mniejsze czasy obliczeń dla użycia `C++ Eigen`.

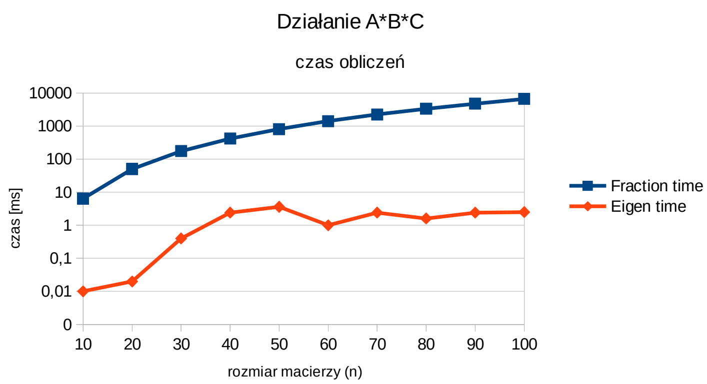


Wykres 1



Wykres 2

3.1.3. Typ `fraction` – porównanie i czas



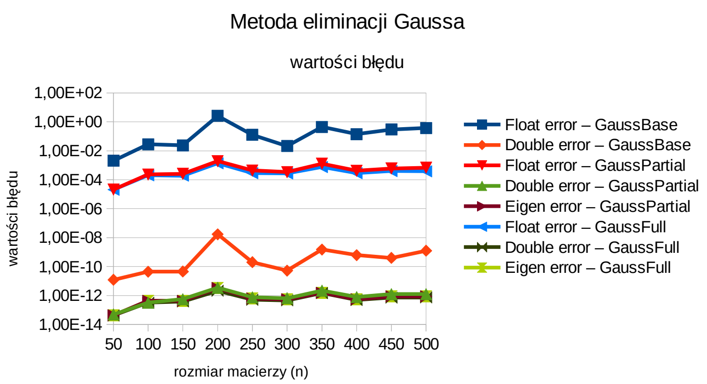
Wykres 3

Wykres 3: W przypadku wykonywania obliczeń na macierzach o precyzji elementów wyrażonej typem `fraction`, w zakresie porównania do wyniku obliczonego przez `C++ Eigen`, wyniki pokrywały się idealnie – błąd wynosił 0. Jedyna różnica to dłuższy czas oczekiwania na wynik. Wniosek przedstawiono na wykresie na przykładzie operacji A*B*C, która z uwagi na największą ilość wykonywanych operacji mnożenia, trwa najdłużej.

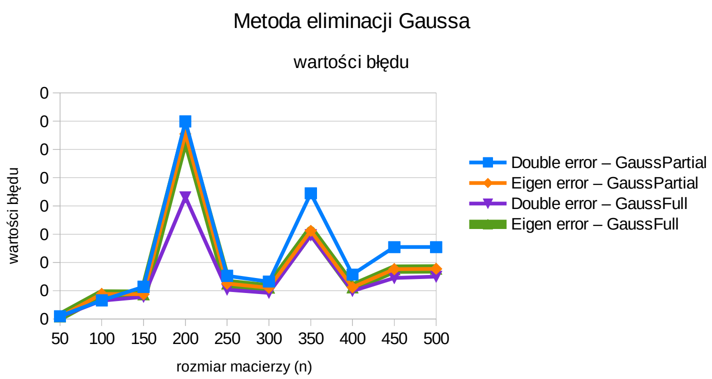
3.2. Rozwiązywanie układów równań liniowych

3.2.1. Typy `double i float` – porównanie i czas

Wykres 4: Przedstawia różnice obliczonego układu dla oryginalnego i wyliczonego wektora (patrz: metoda 1). Im niższa wartość w danym punkcie, tym precyzyjniejszy jest wynik. Najprecyzyjniejszym okazuje się wynik wyliczony przez `C++ Eigen`. Na wykresie 4a zaprezentowano powiększenie obszaru, w którym wartości błędu dla poszczególnych metod obliczania są do siebie zbliżone.

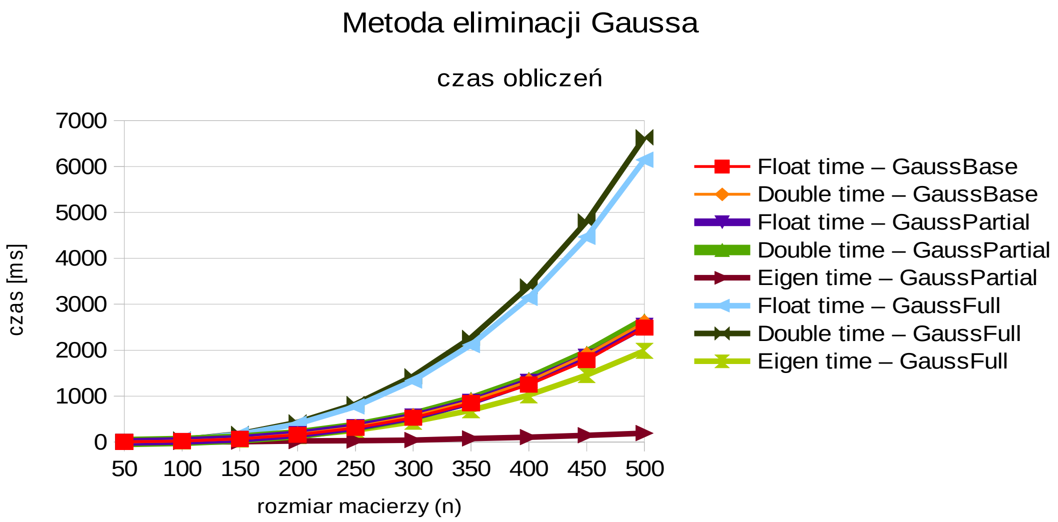


Wykres 4



Wykres 4a

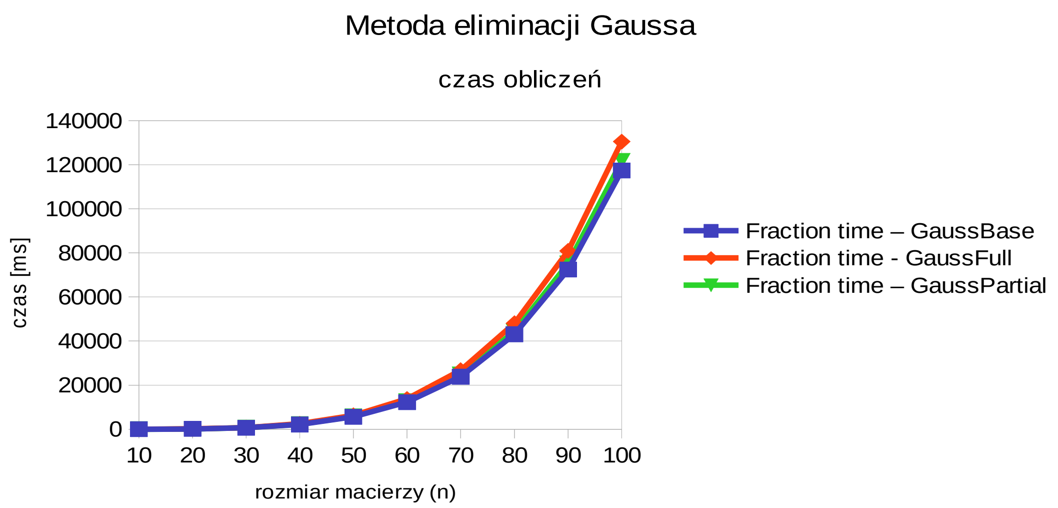
Wykres 5: Przedstawia czas potrzebny do obliczenia rozwiązania układu.



Wykres 5

3.2.2. Typ `fraction` – porównanie i czas

Wykres 6: Przedstawia czas potrzebny do wyliczenia wyników w typie `fraction`, o idealnej precyzji. Z uwagi na idealną precyzję typu, błąd jest zawsze równy 0, więc oszczędzono pokazywania różnic (a właściwie ich braku) na wykresie. Wraz ze wzrostem rozmiaru macierzy, czas oczekiwania na wynik bardzo szybko rośnie.



Wykres 6

4. Szczegóły wykonania zadania

Podział obowiązków	
Barzowska Monika	Bienias Jan
Generowanie losowych macierzy w `C++`	Pobieranie wyników `z C++` do `C#`
Generowanie wyniku z biblioteki `Eigen3` w `C++`	Implementacja eliminacji Gaussa do klasy `MyMatrix`
Implementacja klasy `Fraction` w `C#`	Implementacja operacji na macierzach
Generowanie wyników końcowych, opracowanie wyników, sporządzenie sprawozdania	Testy poprawnościowe