Sprawozdanie - zadanie 2

Krzysztof Kulewski, 238149, grupa 2, 19.11.2017

Opis badania

Badanie polegało na porównaniu błędów i czasu działania wybranych operacji na macierzach, stosując:

- wydajną bibliotekę Eigen 3 (C++),
- własną implementację w wybranym języku (C#).

Wygenerowano trzy losowe macierze (A, B, C) o liczbie wierszy i kolumn równej 50, oraz wektor X, z taką samą liczbą kolumn. Następnie przeprowadzono na nich poniższe operacje:

- $A \cdot X$,
- $(A + B + C) \cdot X$,
- $A \cdot (B \cdot C)$,

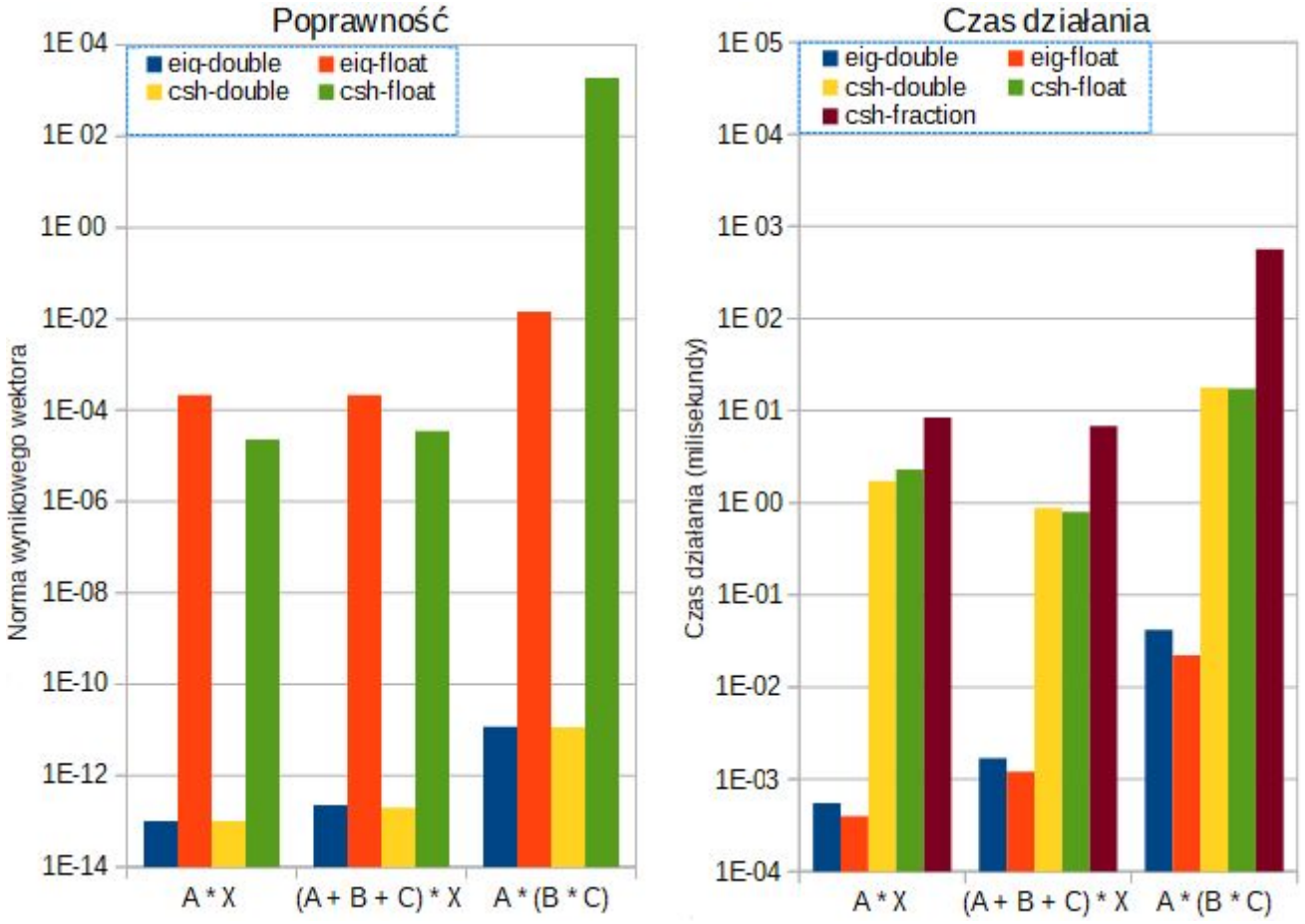
oraz rozwiązano układ równań $A \cdot Y = X$ w trzech wariantach:

- bez wyboru elementu podstawowego (no pivot),
- z wyborem częściowym (partial pivot),
- z wyborem pełnym (full pivot).

Operacje zostały wykonane tysiąc razy i użyto w nich typów *float* oraz *double*.

Przy wyliczaniu norm wektorów i macierzy, punktem odniesienia był *własny typ*, który przechowuje liczbę w postaci ułamka liczb całkowitych typu BigInteger, umożliwiający uniknięcie utraty precyzji.

Dodawanie i mnożenie



Analiza i wnioski:

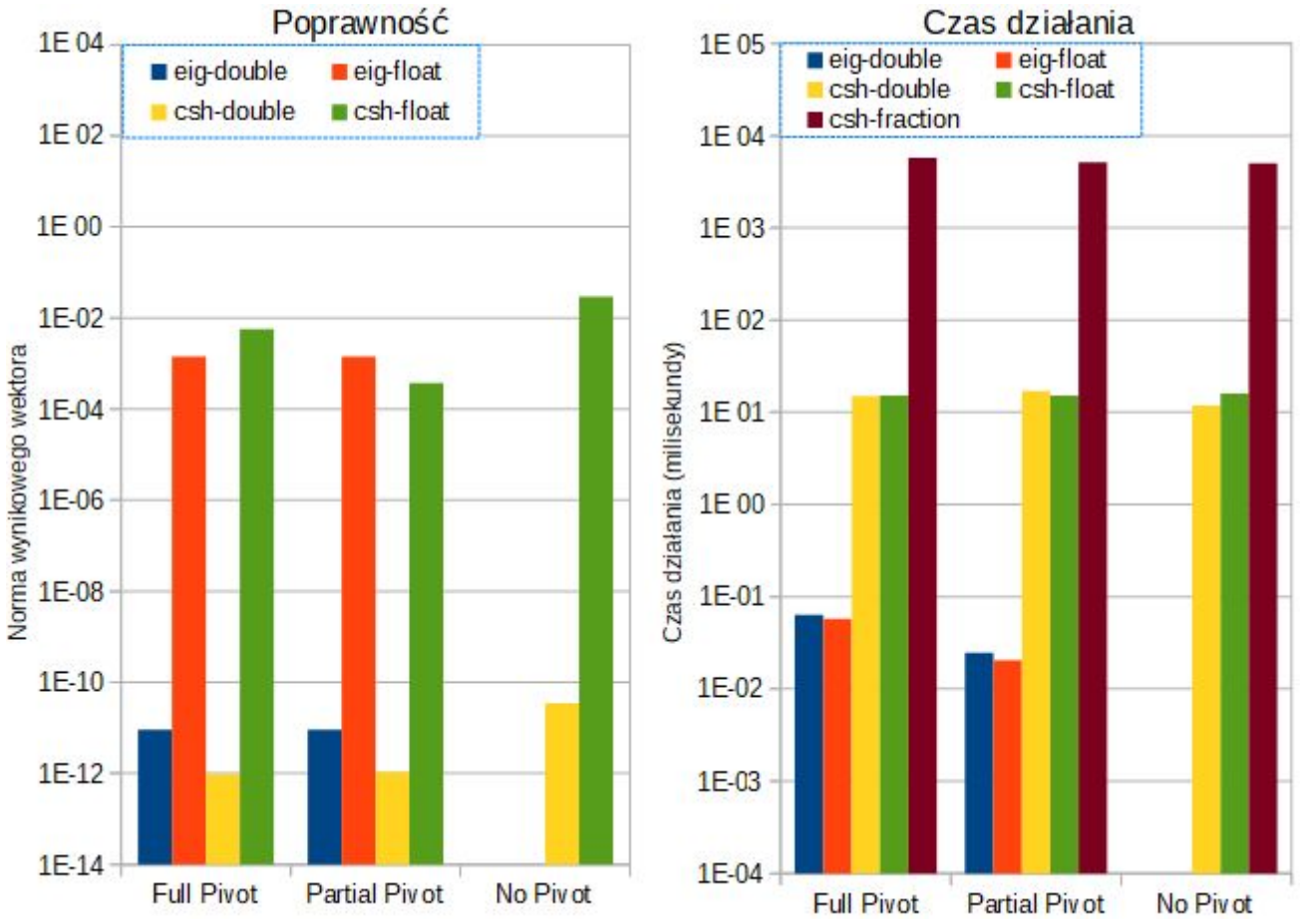
Dla typu *double*, dokładność uzyskanych wyników jest relatywnie dobra, tj. oddalona o dwa rzędy wielkości od rzeczywistej precyzji tego typu w okolicy 0.
Tyczy się to zarówno biblioteki Eigen 3, jak i naiwnej implementacji.

Zastosowanie typu *float* znacząco pogarsza poprawność wyników, gdyż dużą rolę odgrywają błędy obcięcia. Szczególnie widoczne jest to w przypadku mnożenia trzech macierzy, gdzie błędy te są wielokrotnie kumulowane. Naiwna implementacja dodatkowo potęguje ten efekt.

W każdym przypadku, czas działania funkcji z biblioteki Eigen 3 był znacznie krótszy od ich odpowiedników w implementacji naiwnej.
Wpływają na to zarówno optymalizacje w samej bibliotece, jak i to, że C#, jako język znacznie wyższego poziomu, opiera się na działaniu Garbage Collectora i jest tłumaczony na kod maszynowy “w locie” (JIT). Ponadto, opakowywanie i rozpakowywanie obiektów powoduje dodatkowy narzut.

Czas działania dla *własnego typu ułamkowego* jest stosunkowo wysoki, ponieważ operacje wykonywane są na bardzo dużych liczbach.

Rozwiązywanie układów równań liniowych - metoda eliminacji Gaussa



Analiza i wnioski:

Dla typu *double*, wyniki uzyskane za pomocą naiwnej implementacji mają lepszą dokładność, niż gdy użyto biblioteki Eigen 3. W przypadku typu *float*, wyniki są porównywalne.

Przyczyną takiego stanu rzeczy może być to, iż punktem odniesienia użytym do obliczenia normy jest *własny typ ułamkowy*, korzystający z naiwnej implementacji. Implementacja ta może być obarczona błędem.

Zgodnie z intuicją, czas działania metody z pełnym wyborem elementu podstawowego jest najdłuższy. W przypadku implementacji naiwnej, metoda z wyborem częściowym była szybsza, niż wersja bez wyboru.

Analogicznie do operacji mnożenia i dodawania macierzy, również tutaj czas działania dla własnego typu ułamkowego był wysoki z powodu działania na bardzo dużych liczbach.

Co więcej, w przypadku większych macierzy (100 x 100 i więcej), uzyskane liczby były zbyt duże, by dało się je podzielić i rzutować na typ *double*.
Skutkowało serią NaN (Not a Number) w wektorze wynikowym.