Algorytmy numeryczne

Zadanie 2
Aleksander Kosma
grupa 1 tester-programista

8 Listopad 2017

1 Dodawanie i Mnożenie

// czy jezykz duzej czy malej Opracowanie prezentuje obliczenia na macierzach, przy użyciu języka Java i wsparciu Eigena wykorzystanego w C++. Pierwsza część opisuje czas działania i błędy operacji dodawania i mnożenia. Obliczenia wygenerowane w Javie porównywane były z wynikami wygenerowanymi z Eigena. Testy zostały przeprowadzone na 3 wzorach:

$$A * X$$

$$(A + B + C) * X$$

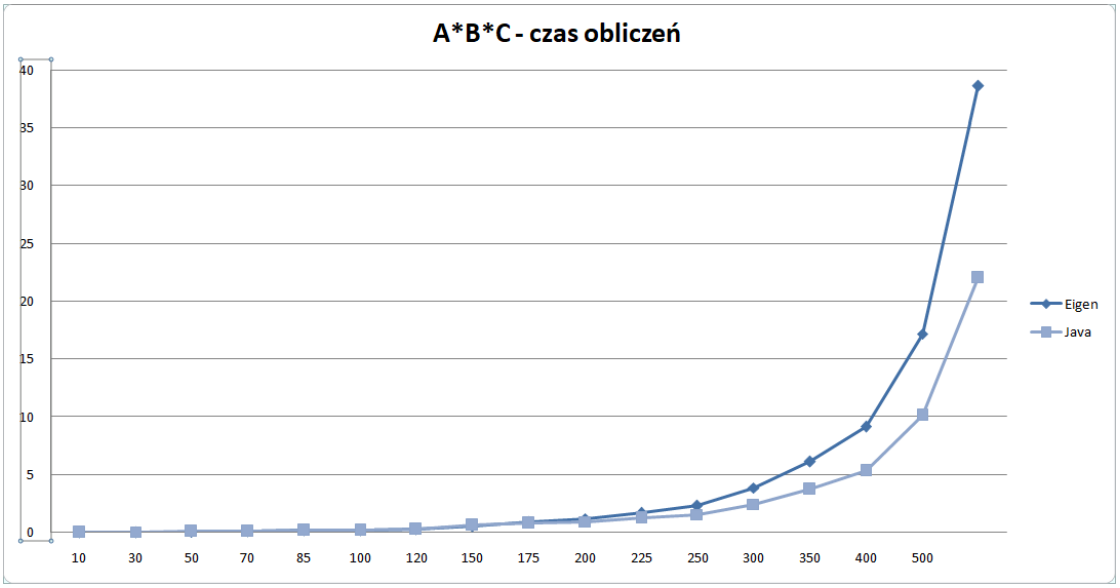
$$A * B * C$$

Gdzie litery A,B,C to macierze, X to wektor.

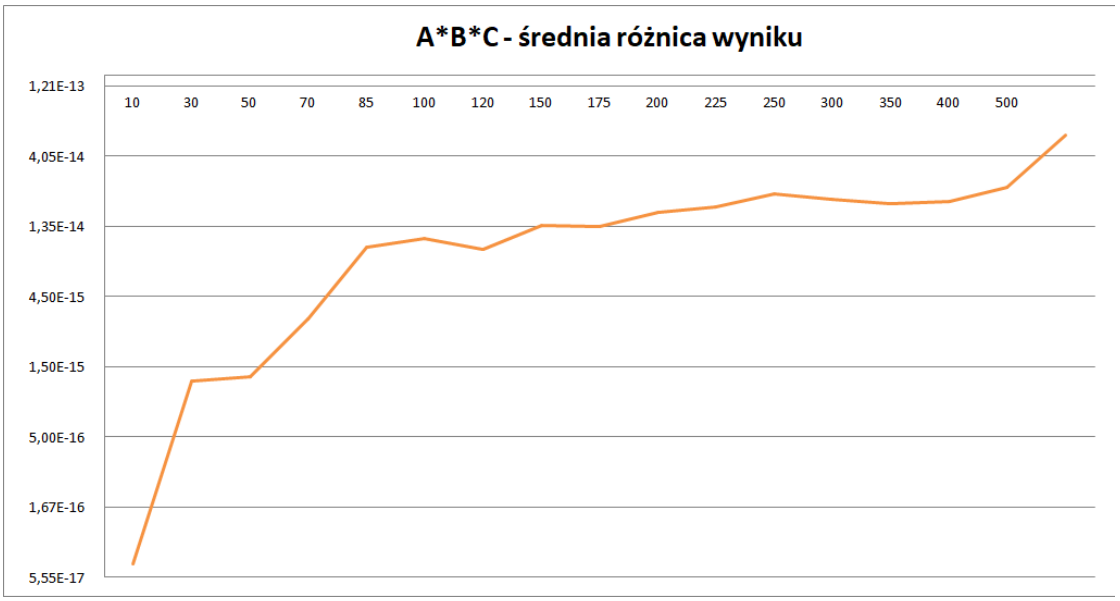
Testy zostały przeprowadzone na 3 typach zmiennych. Double, Float i na własnym typie, który przechowuje liczbę w postaci ułamka. Typ ten ma bezstratną precyzję.

Wykresy obliczone dla Double:

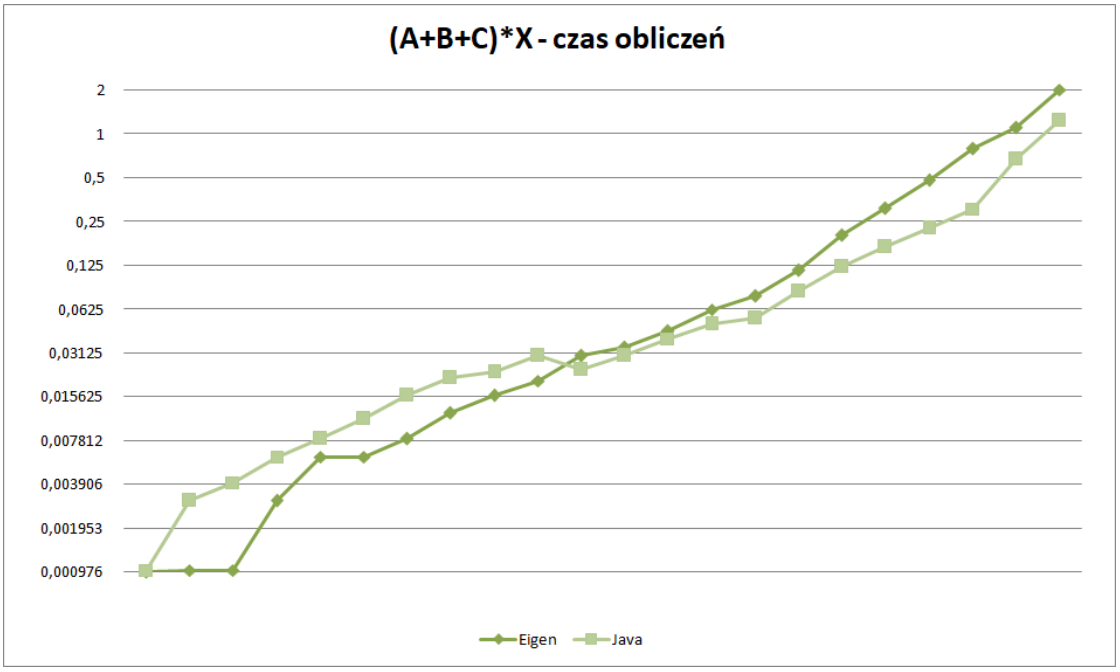
Pierwszy wykres prezentuje czas potrzebny do obliczenia podanego wzoru. Pod koniec widać przewagę własnych obliczeń nad Eigen.



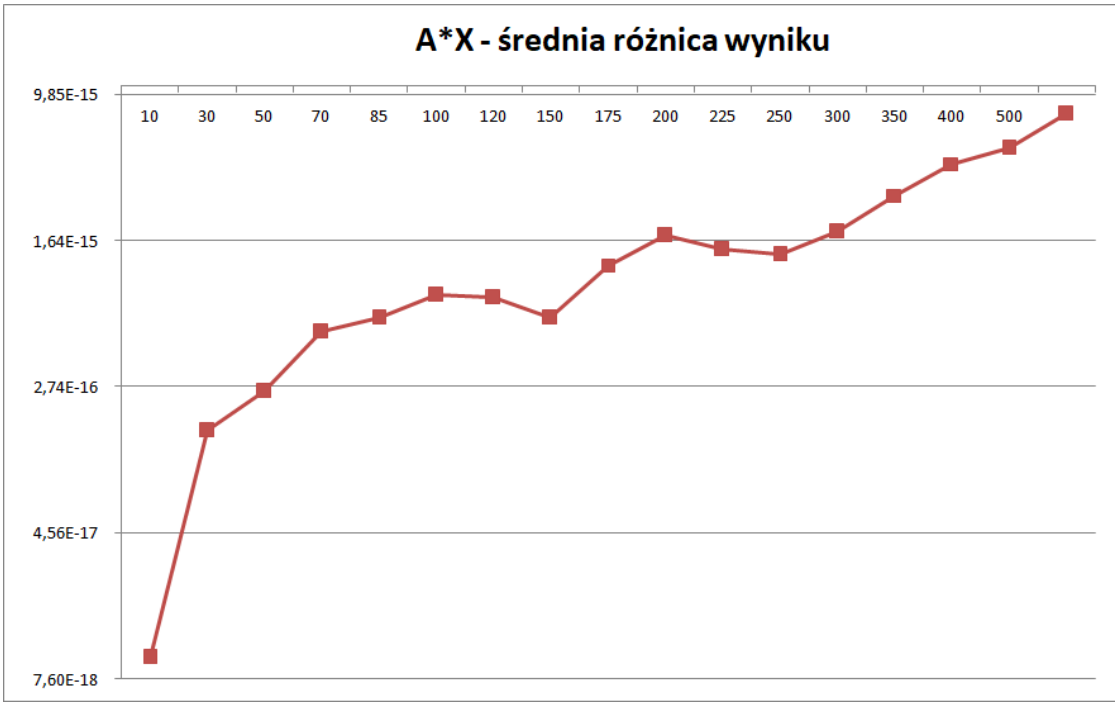
Drugi wykres prezentuje te same dane, tylko w kontekście precyzji. Widać ze wraz ze wzrostem wielkości macierzy, błędy przybierają na wartości.



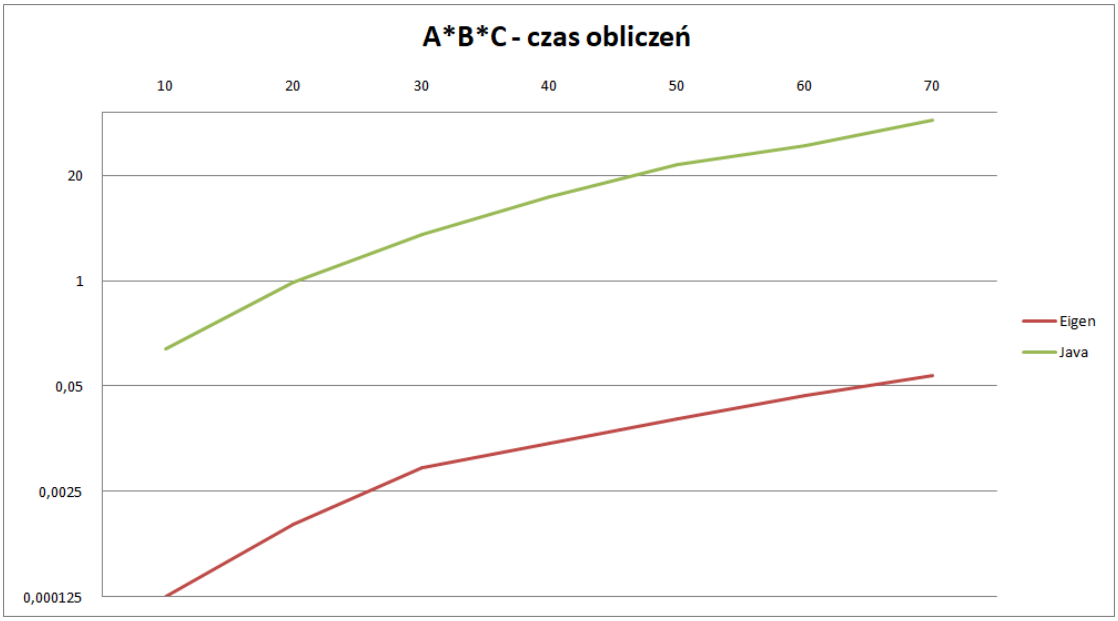
Trzeci wykres prezentuje dla innego wzoru czas obliczeń. W pewnym momencie obliczenia Javy stają się szybsze i tendencja ta się utrzymuje.



Czwarty wykres prezentuje podobne wnioski jak w przypadku pierwszego. Tym większa macierz tym większy błąd ostateczny.



Obliczenia na Float wykazywały podobną tendencje, tyle że w mniejszej skali i wolniejszym tempie, więc pominę prezentacje dla tej zmiennoprzecinkowej. W przypadku Własnej precyzji, w zakresie Double wyniki pokrywały się idealnie. Główna różnica to dłuższy czas oczekiwania na wynik.



Drugi wykres prezentuje bezwzględną różnice między wynikiem biblioteki a sumowaniami. Można zauważyć, że przez większą część spektrum wartości, sumowanie tylne jest precyzyjniejsze nawet o 2 rzędy wielkości.

Takie rezultaty wynikają z natury działania zmiennej double. W przypadku sumowania liczby większej do liczby mniejszej(gdzie różnica wynosi parę rzędów wielkości), zmienna odrzuca końcowe cyfry mniejszej liczby by pomieścić najistotniejsze wartości. Stąd utrata precyzji. W momencie kiedy dwie liczby mają podobną wartość, ich suma potrzebować będzie podobnej precyzji do zapisania wyniku. W konsekwencji nie odrzuci danych. Podsumowując kwestie precyzji wyniku. Opierając się o zmienną double, można stwierdzić, że precyzja wyniku waha się między 14, a 16 miejscem po przecinku.
Na obrazku poniżej znajduje się praktyczny przykład. Dla uproszczenia zagadnienia jest on prezentowany na zmiennej float(32 bity) która zazwyczaj mieści w sobie 8 cyfr dziesiętnych.

By rozwiązać problem niskiej precyzji, który zaistniał w przypadku doubla, trzeba mieć kontrolę nad precyzją obliczeń. Rozwiązaniem może okazać się klasa BigDecimal. Klasa ta może przechować nieograniczoną wartość liczbową, ogranicza ją jedynie skala która osiąga wartość 32-bitowego integera, czyli trochę ponad dwa miliardy miejsc.
Udało mi się osiągnąć praktycznie nieograniczoną precyzję. Wyniki oparte o BigDecimal porównywałem z kalkulatorem online Wolfram(computational knowledge engine). Przykładowy wypis programu wraz ze wszystkimi rodzajami obliczeń i ich wynikami. W tym przypadku podana precyzja BigDecimal i Wolfram to 64 miejsca po przecinku.

2 Rozwiązywanie układów równań liniowych

Druga część opisuje rozwiązywanie układu równań liniowych macierzy A i wektora B. W Javie wykorzystane zostały 3 wersje metody eliminacji Gaussa: -bez wyboru elementu podstawowego -z częściowym wyborem elementu podstawowego -z pełnym wyborem elementu podstawowego Odnośnikiem jako poprawny wynik były wektory obliczone przez Eigen. Eigen ma możliwość oblicznia układu poprzez częściowy wybór, albo pełny elementu podstawowego.