Algorytmy numeryczne

Zadanie 2

Aleksander Kosma Tomasz Adamczyk
grupa 1 tester-programista

8 Listopad 2017

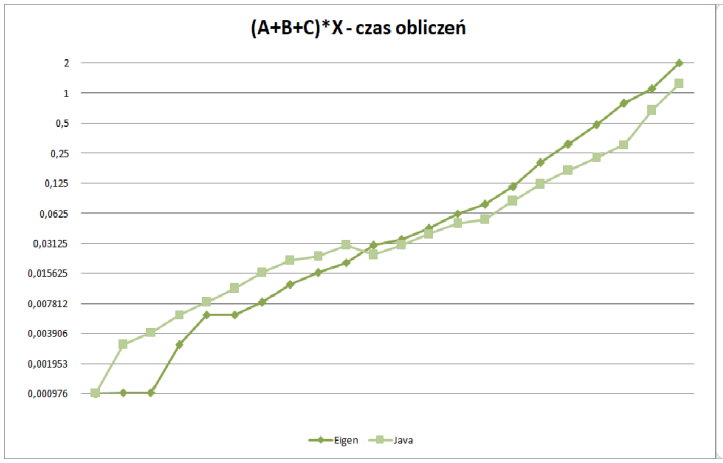
1 Dodawanie i Mnożenie

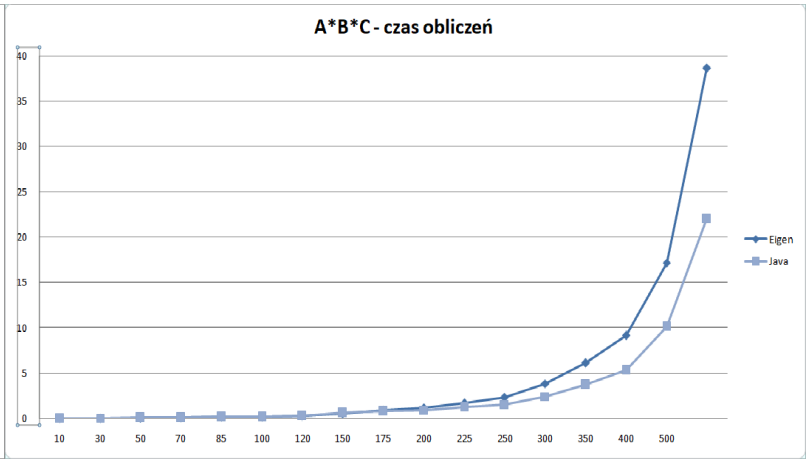
Opracowanie prezentuje obliczenia na macierzach, przy użyciu języka java i wsparciu biblioteki eigen wykorzystanej w C++. Pierwsza część opisuje czas działania i błędy operacji dodawania i mnożenia. Obliczenia były generowane w javie porównywane były z wynikami wygenerowanymi z eigena. Testy zostały przeprowadzone na 3 wzorach:

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

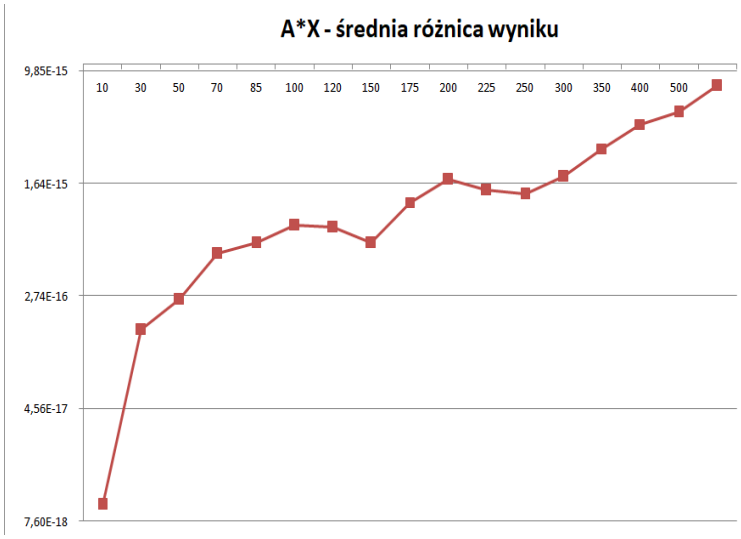
Gdzie litery A,B,C to macierze, X to wektor. Testy zostały przeprowadzone na 3 typach zmiennych. Double, Float i na własnym typie, który przechowuje liczbę w postaci ułamka. Typ ten ma bezstratną precyzję. Liczby które rozstały wykorzystane do obliczeń mieściły się w przedziale od 0 do 1, w postaci dziesiętnej, z wykorzystaniem precyzji double.

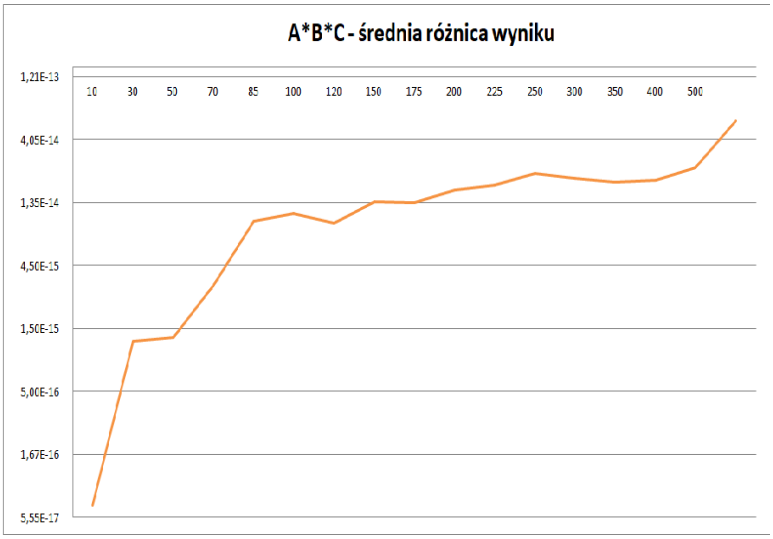
Pierwsze dwa wykresy prezentują czas potrzebny do obliczenia podanego wzoru. Os pionowa reprezentuje czas w sekundach, oś pozioma rozmiar macierzy. Wraz ze wzrostem rozmiaru macierzy, we wszystkich przypadkach(odnotowane w opracowaniu tylko dwa) widać czasową przewagę własnych obliczeń nad eigen.Podane wykresy utworzone na zmiennej double.



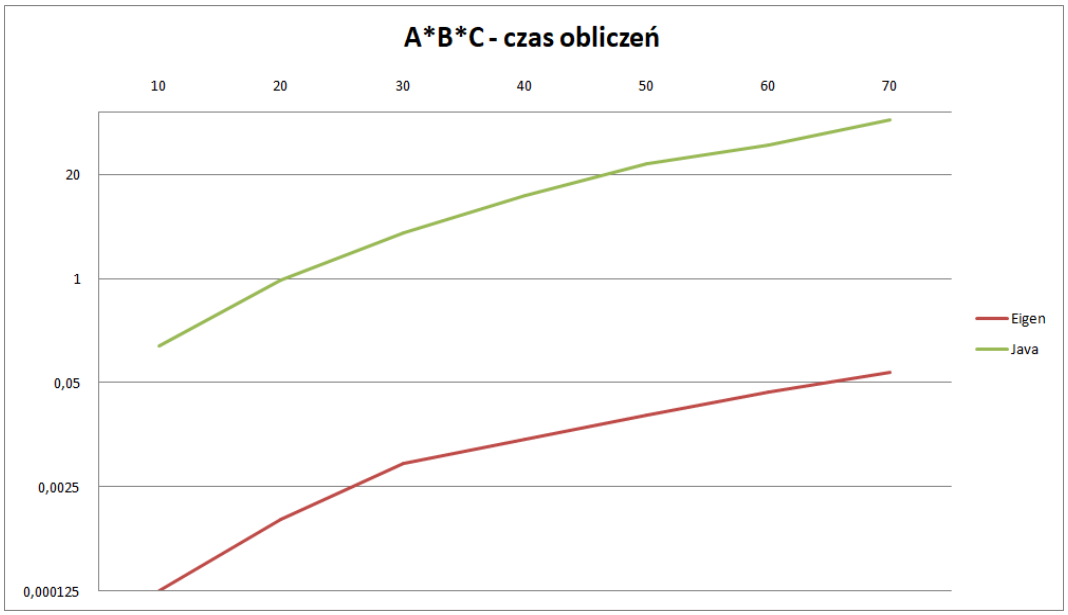


Kolejne dwa wykresy odnotowują uzyskaną precyzje. Wyznaczona ona była poprzez wyznaczenie średniej wartości dla końcowej macierzy i odjęcie od siebie wyniku z eigena i javy. Widać ze wraz ze wzrostem wielkości macierzy, błędy przybierają na wartości.Tym większa macierz tym większy błąd ostateczny. W przypadku mnożenia wynika to z faktu obliczania kolejnych miejsc po przecinku, które nie mieszczą się już w zakresie danej liczby zmiennoprzecinkowej.





Obliczenia na Float wykazywały podobną tendencje, tyle że w mniejszej skali i wolniejszym tempie, więc pominięto prezentacje dla tej zmiennoprzecinkowej. W przypadku własnej precyzji, w zakresie porównania do wyniku obliczonego przez Eigen, wyniki pokrywały się idealnie. Główna różnica to dłuższy czas oczekiwania na wynik.



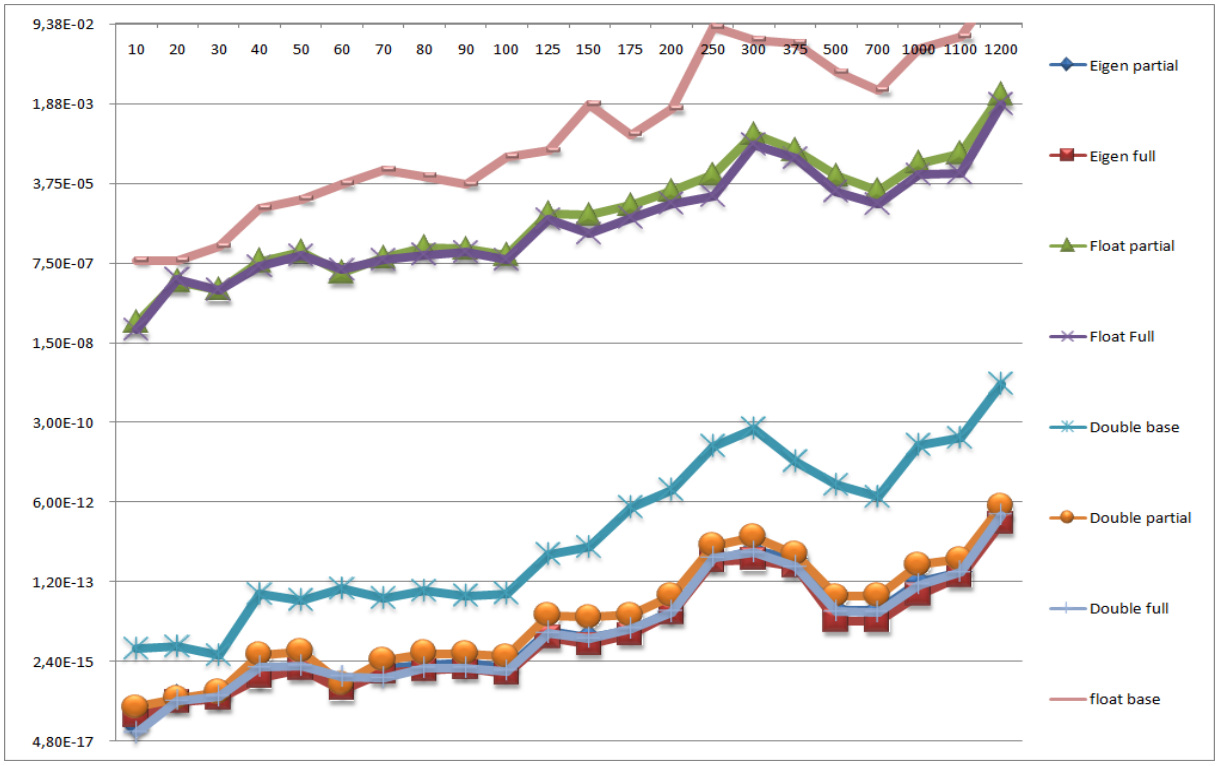
2 Rozwiązywanie układów równań liniowych

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

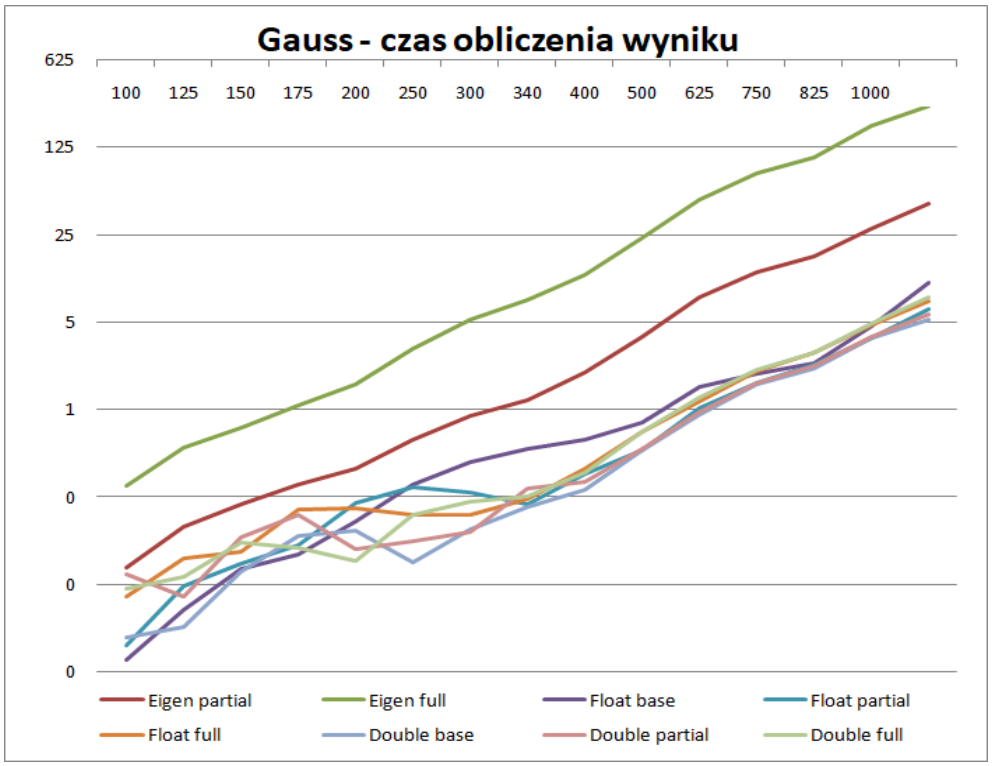
Odnośnikiem jako poprawny wynik było podstawienie pod układ obliczonych niewiadomych. W eigen skorzystano z wyliczenia poprzez częściowy i pełny wybór.

Pierwszy wykres przedstawia różnice obliczonego układu dla wyliczonych niewiadomych i oryginalnego wektora.Tym niższa wartość w danym punkcie tym precyzyjniejszy wynik.Najlepszy okazuje się wynik wyliczony przez Eigen. Niewiele gorzej wypada Double dla pełnego wyboru elementu podstawowego.

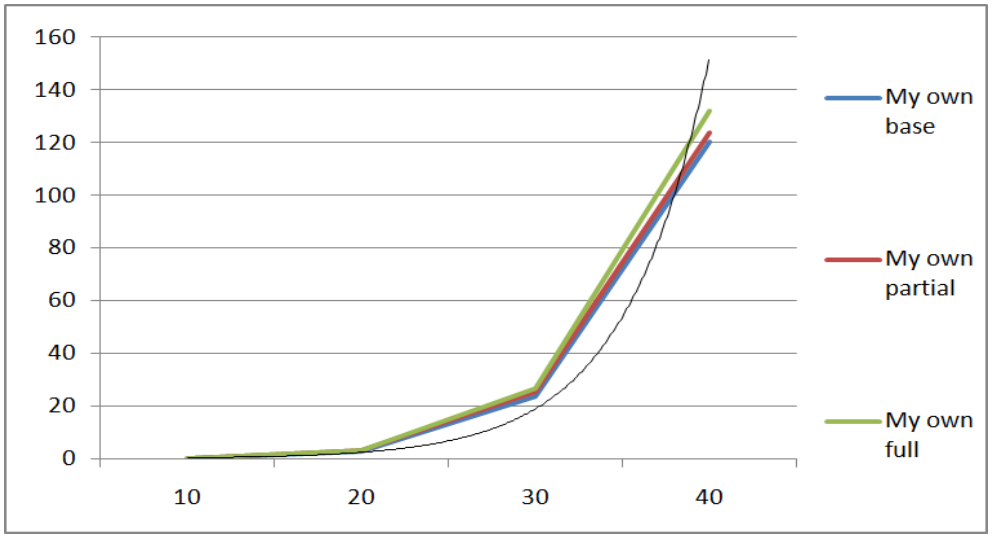


.

Drugi wykres przedstawia czas potrzebny do osiągnięcia rozwiązania. Chaotyczny rozkład i przenikanie przez siebie dolnych linii wynika z sytuacji, gdzie dana macierz może mieć odpowiedni układ nie wymagający jakiegokolwiek sortowania. Stąd tak zbliżone wyniki.



Ostatni wykres przedstawia czas potrzebny do wyliczenia wyników dla idealnej precyzji. W istocie wynik wychodzi za każdym razem idealny. Wraz ze wzrostem macierzy, czas oczekiwania na wynik bardzo szybko rośnie co widać na dorysowanej na szaro funkcji.



Większość obliczeń została uśredniona z 5-15 próbek na każdy rozmiar macierzy. Zazwyczaj tyle wystarczało by zauważyć tendencje wzrostowe.

Podział obowiązków	
Aleksander Kosma	Tomasz Adamczyk
- Klasa MyOwnPrecision	-Implementacja Gaussa do Klasy MyMatrix
- Szkielet klasy i zapewnienie generyczności	- Klasa do generowania losowych macierzy
- Implementacja metod dodawania i mnożenia	- Generowanie wyniku z biblioteki Eigen w C++
- Testy i generowanie wyników	- Zautomatyzowanie generowania wyników
- Obróbka wyników, wykresy i opracowanie	-