Krzysztof Borawski 238152 grupa 1 tester-programista 17.11.2017
Sprawozdanie z zadania 2
“Operacje na macierzach”

1. Dodawanie oraz Mnożenie macierzy

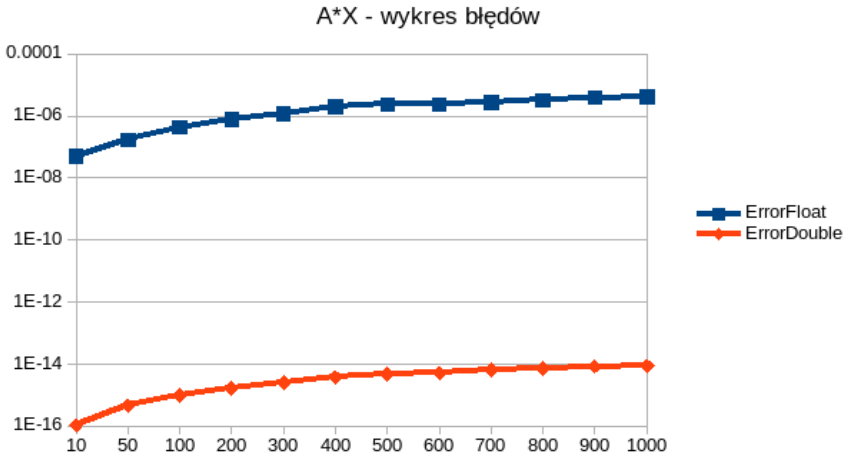
Obliczenia na macierzach zostały wykonane na dwóch typach danych Float oraz Double przy użyciu jezyka C++ oraz wsparciu biblioteki Eigen. Wartości macierzy były generowane losowo przy pomocy metody “Random()”. Następnie wartości te były przekazywane do wektorów wewnątrz klasy “MyMatrix” na których wykonywane zostały operacje. Testy zostały przeprowadzone dla trzech wzorów:

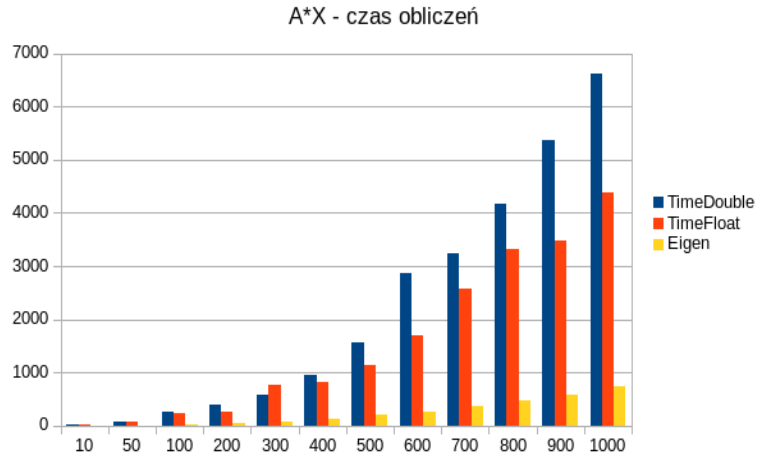
$$A * X \qquad (A + B + C) * X \qquad A * (B * C)$$

Gdzie X jest wektorem a A, B, C to macierze kwadratowe. Celem testów było zbadanie błędów oraz wydajności tych operacji. Wydajność została mierzona przy pomocy standardowej biblioteki C++ <chrono>.

1.1 *A * X*

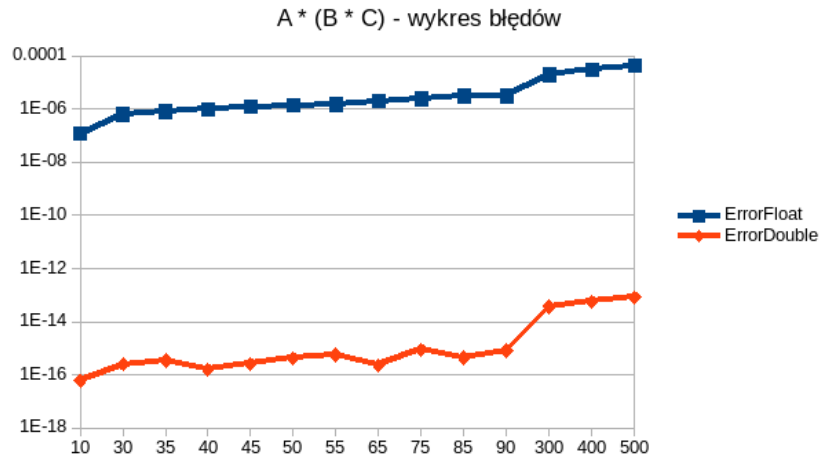
Pierwszy wykres reprezentuje średnie wartości błędu dla operacji A*X. Są to uśrednione wartości błędów bezwzględnych miedzy własnymi operacjami a wynikami z biblioteki eigen. Oś pionowa reprezentuje wielkość błędu a oś pozioma wielkość macierzy. Wykres drugi reprezentuje czas potrzebny do wykonania operacji mierzony w mikrosekundach.

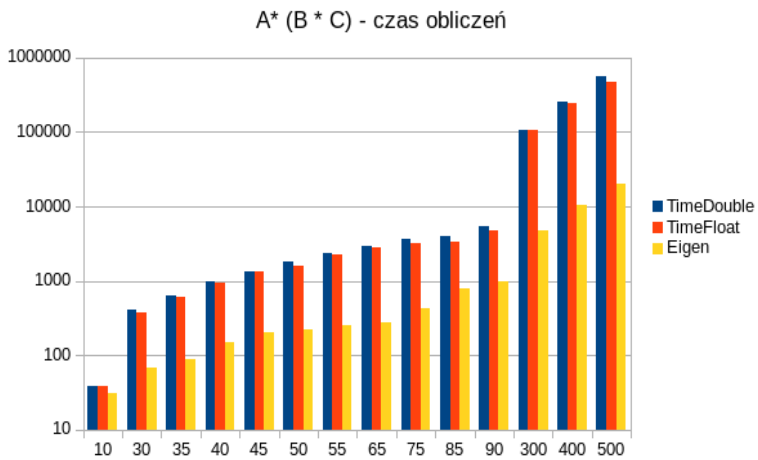




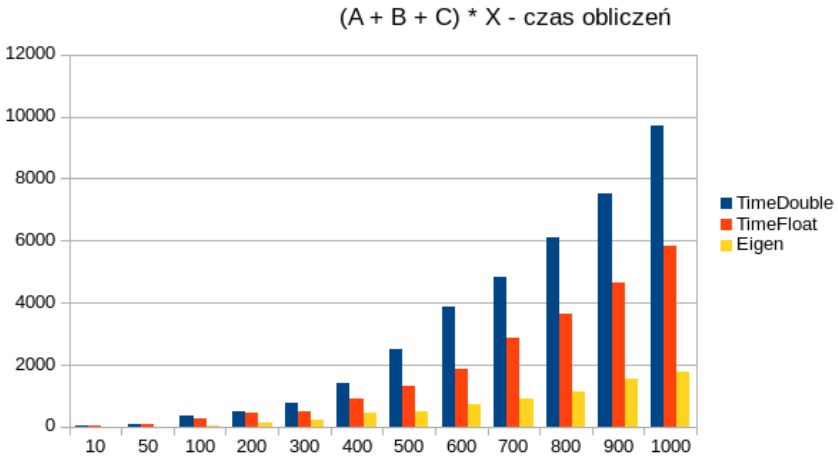
Można zaobserwować, że wraz ze wzrostem rozmiaru macierzy błedy wraz z czasem rosną na wartości. Można również zaobserwować przewage eigena nad własnymi obliczeniami. W pozostałych dwóch przypadkach istnieją podobne tendencje.

1.2 *A * (B * C)*





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



Główną różnica miedzy *A * X* a *(A + B + C) * X* jest czas potrzebny do wykonania obliczeń dlatego wykres błedow został pominięty.

1.4 Tabela porównująca *A * X* oraz *A * (B * C)*

Poniższa tabela reprezentuje różnice wyników dla operacji mnożenia macierzy przez wektor a mnożenia macierzy przez macierz. Dane dla macierzy rozmiaru 500x500 typu double.

	MaxError	**MinError**	**AverageError**	**TimeNeeded**
A * X	3.21E-14	0	4.73E-15	1571
A * (B * C)	7.96E-13	0	8.37E-14	549336

Najbardziej rzucającym się w oczy jest fakt ogromnego wzrostu czasu potrzebnego do obliczenia operacji mnożenia trzech macierzy kwadratowych.

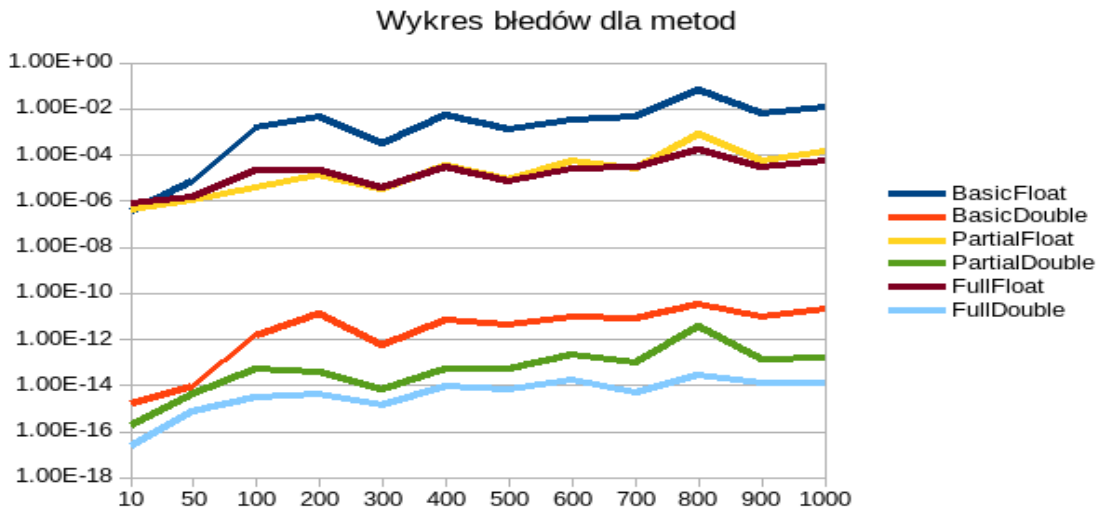
2. Rozwiązywanie układów równań liniowych.

Poniższe tabele opisują rozwiązanie układu równań linowych macierzy A oraz wektora X. Zaimplementowane zostały trzy metody eliminacji Gaussa:

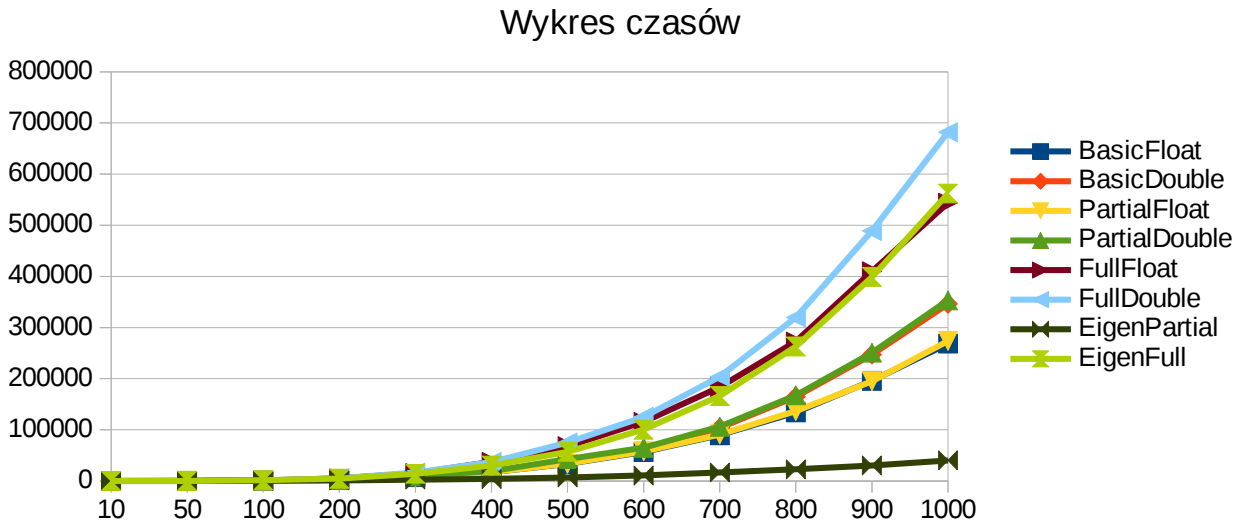
- Gauss bez wyboru el. Podstawowego (basic)
- Gauss z częściowym wyborem el. Podstawowego (gaussPartialPiv)
- Gauss z całkowitym wyborem el. Podstawowego (gaussFullPiv)

Do porównania zostały wykorzystane dwie metody z biblioteki Eigen:

- partialPivLu()
- fullPivLu()



Drugi wykres przedstawia czas potrzebny do osiągnięcia rozwiązania. Po raz kolejny oś pionowa przedstawia czas w mikrosekundach a oś pozioma rozmiar macierzy. Przenikanie sie przez siebie wartości przy macierzach małych rozmiarów może być spowodowany faktem, że dany układ mógł nie wymagać wielu zamian.



2.1. Tabela porównująca czasy oraz rozmiary błędów dla macierzy rozmiaru 1000x1000

	Average Error	Time
Basic Float	1.12E-02	268672
Basic Double	2.00E-11	346485
Partial Float	1.36E-04	273822
Partial Double	1.57E-13	353408
Full Float	5.13E-05	543889
Full Double	1.18E-14	681745
Eigen Partial	+	40428
Eigen Full	+++	562039

3. Wnioski

- Błedy narastają wraz ze wzrostem macierzy.
- Metoda wyboru całkowitego elementu podstawowego jest metodą wyliczającą z najwiekszą dokładnością kosztem najdłuższego czasu.
- Metoda gaussa bez wyboru jest metodą najszybszą (poza obliczeniami eigena) aczkolwiek licząc tą metodą otrzymamy największe błędy.
- Metoda częściowego wyboru jest dobrym kompromisem miedzy obiema metodami ponieważ czas obliczeń nie jest tak długi jak w przypadku całkowitego wyboru a dokładność obliczeń jest o wiele wyższa niż w przypadku zwykłego gaussa.
- Metody dla float liczą szybciej niż dla double.
- Mimo, że nie wykonałem obliczeń dla klasy reprezentującej ułamek domyślam się, że czas operacji dla ułamków powinien rosnąć w przerażająco dużym tempie

Sposób kompilacji : g++ MyMatrix.cpp -O3 -march=native
Procesor : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Architektura: x86_64