Gdańsk, 9 Grudnia 2018

# Algorytmy Numeryczne – Zadanie 3

## Protokoły populacyjne

Andrzej Chorostian
Informatyka 1st. 3r. gr.T1

## 1.Temat

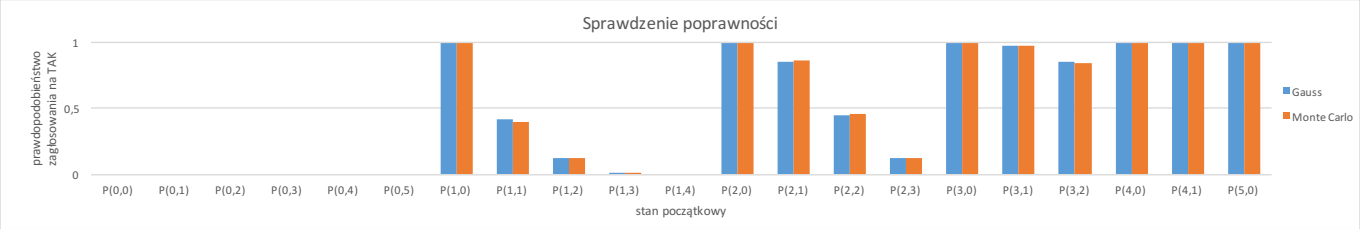
Protokoły populacyjne to model obliczeń stworzony do badania algorytmów rozproszonych. W tym modelu obliczenia są prowadzone przez poruszających się agentów o bardzo ograniczonych możliwościach. Celem zadania jest napisać program, który oblicza prawdopodobieństwo zagłosowania na TAK przy liczbie agentów równej N i określonym stanie początkowym.

## 2.Analiza i specyfikacja

Do napisania programu postanowiłem wykorzystać język C++ korzystając z bibliotek standardowych oraz biblioteki Eigen3, z której wykorzystałem implementację macierzy i wektorów rzadkich. Zaimplementowałem również własną klasę przechowującą macierz rzadką w kilku strukturach std::vector. We wszystkich testach wartości z macierzy przechowywane są w typie double a eliminacja Gaussa występuje tu w wersji z częściowym wyborem elementu podstawowego. Wszystkie algorytmy sprawdzane są poprzez wymnożenie macierzy A z wektorem wynikowym X i porównanie z początkowym wektorem B.

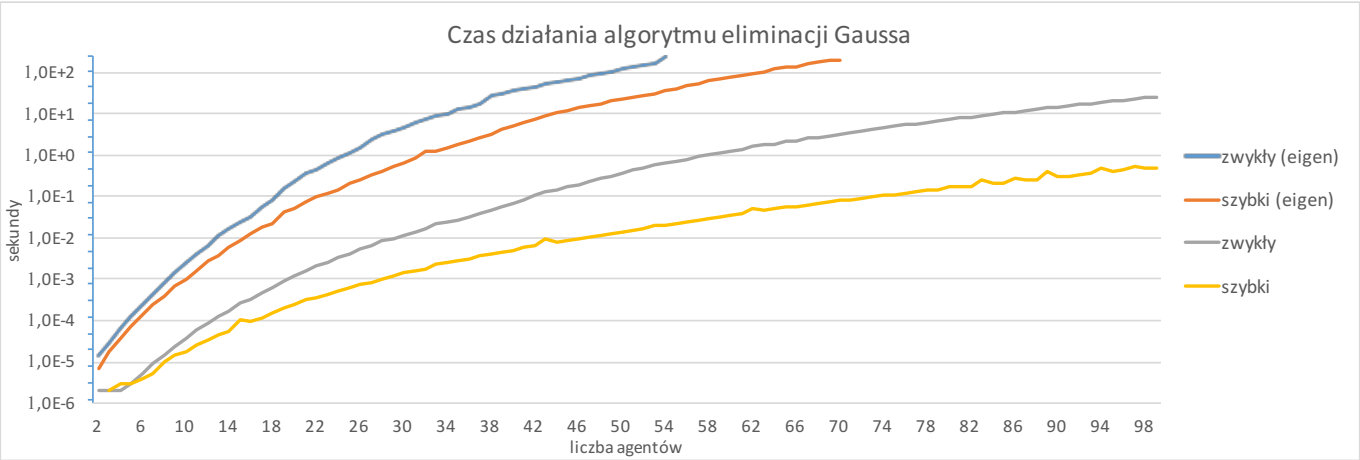
## 3.Poprawność implementacji

Poprawność konstruowania układu równań stwierdziłem porównując wyniki z utworzonej przeze mnie macierzy z wynikami doświadczenia Monte Carlo. Poniżej przedstawiam porównanie efektów dla liczby agentów równej 5.



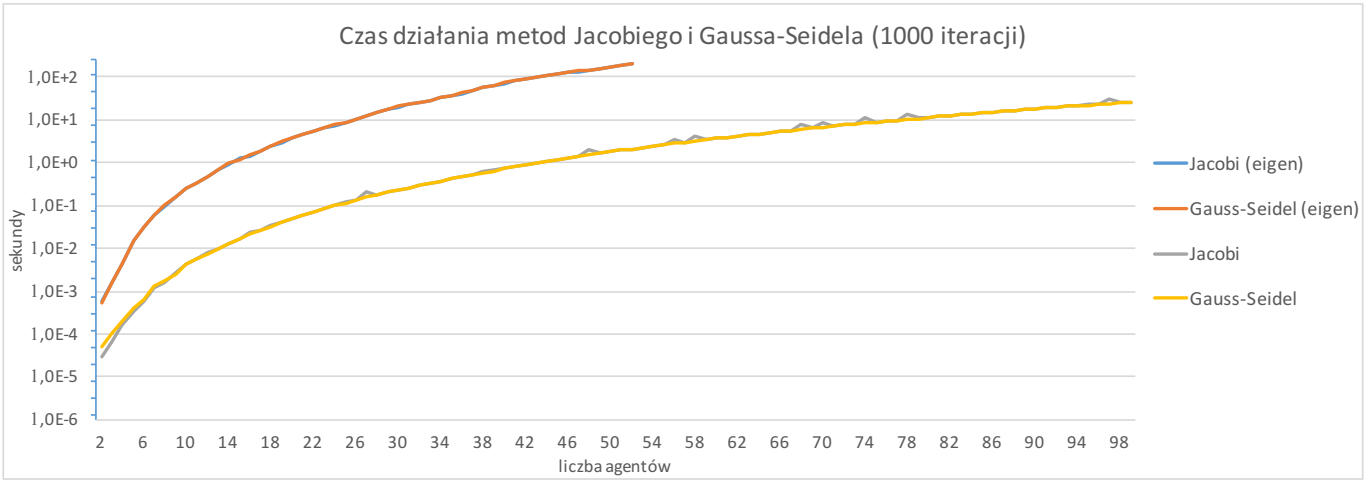

## 4.Usprawnienie eliminacji Gaussa

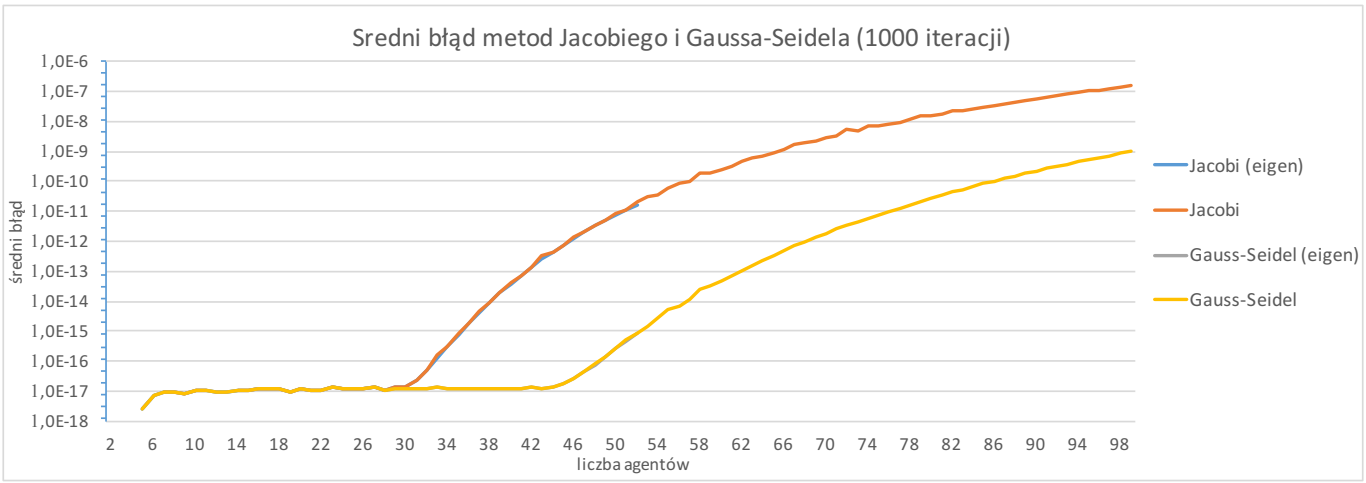
Dla większej liczby agentów powstające układy równań będą miały bardzo dużo zer. Można przyśpieszyć działanie algorytmu poprzez pominięcie mnożenia całego wiersza w celu wyzerowania pierwszego elementu, jeśli jest już równy zeru. Porównanie zwykłego i usprawnionego algorytmu zamieściłem poniżej.



## 5.Metody Iteracyjne

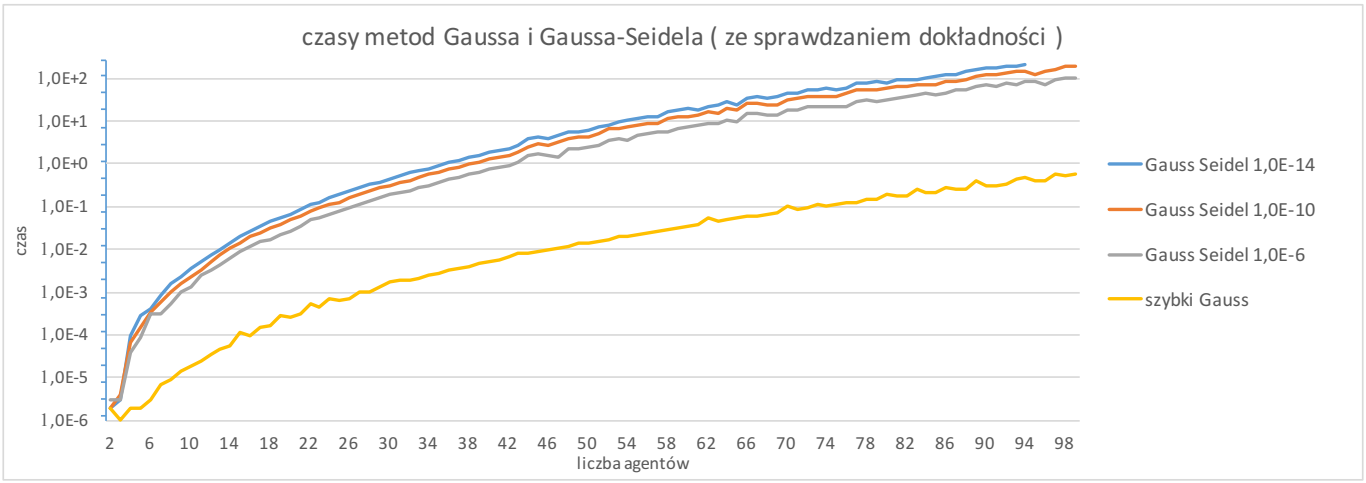
Do rozwiązania układu równań można również stosować metody iteracyjne aby otrzymać wektor wynikowy. Porównałem więc metodę Jaciobiego oraz Gaussa-Seidela. Obie metody wykonały 1000 iteracji. Metoda Jacobiana otrzymane wyniki wykorzystuje dopiero w następnej iteracji, Natomiast Gaussa-Seidela już podczas obliczania kolejnych wartości. Dzięki temu ta metoda jest dokładniejsza, a wykonuje się w praktycznie identycznym czasie. Wyniki testów przedstawione zostały na poniższych wykresach.



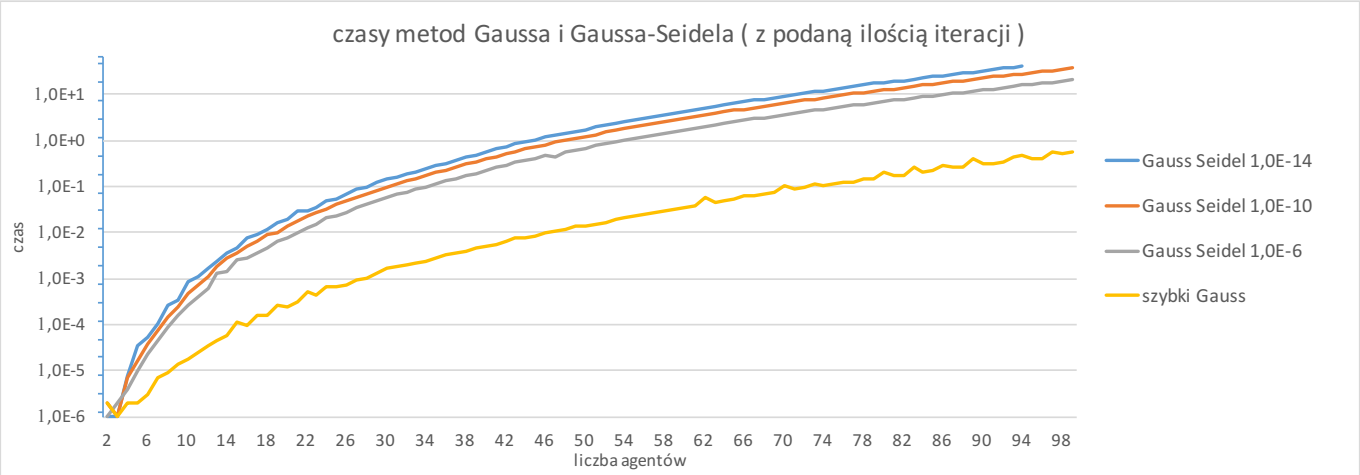




## 6.Dokładność obliczeń

Zakładając, że interesują nas wyniki w konkretnej dokładności, można ograniczyć ilości iteracji, co może skrócić czas potrzebny na wykonanie obliczeń. Tutaj metody iteracyjne wykonują się, dopóki największy błąd w wektorze wynikowym jest mniejszy od żądanej dokładności. Wyniki tych testów prezentują się następująco:

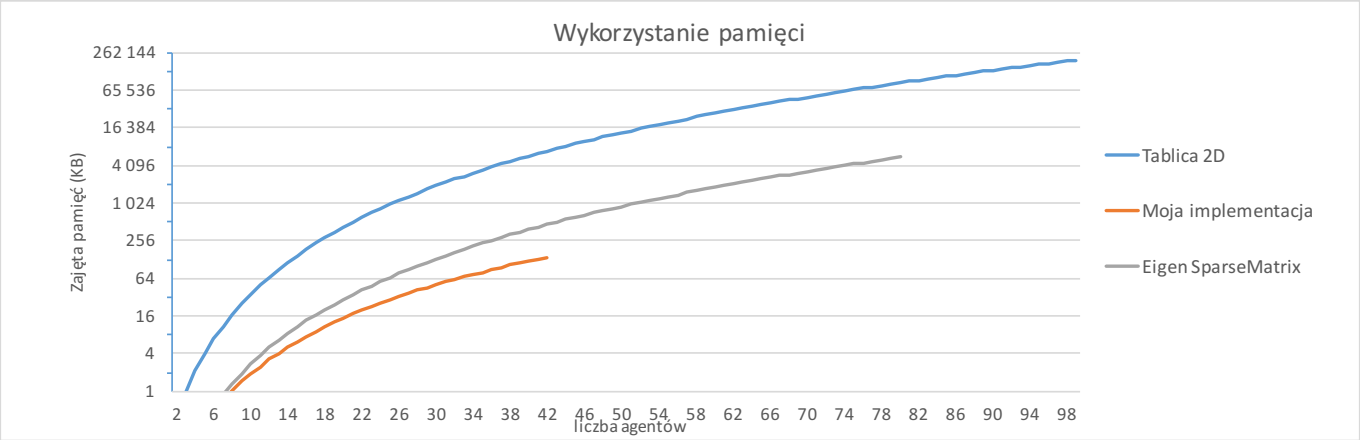


Na podstawie powyższego testu, dla każdego N została obliczona ilość wymaganych iteracji. Jeśli ten algorytm miałby być wiele razy powtarzany, to można by używać wyznaczonej liczby iteracji dla danej dokładności. Wtedy czasy wykonywania tych operacji prezentowałyby się następująco:




## 7.Struktura danych

Jeśli do obliczenia jest naprawdę duża macierz, pamięci podręcznej jest mało a czas obliczeń nie ma dla nas znaczenia, można wykorzystać inne struktury danych, które pomijają zera. Przygotowałem własną implementację takiej struktury opartą na trzech wektorach - dwa odpowiedzialne za zaznaczenie pozycji, a trzeci za przechowanie wartości. To rozwiązanie jest bardzo powolne, jednak oszczędniejsze dla pamięci niż przechowywanie wartości w tablicy. Moje rozwiązanie, porównałem z gotową już implementacją takiej struktury z biblioteki Eigen3.
W poprzednich testach można było zauważyć, jak bardzo wolniejsze jest użycie tej struktury. Teraz czas na porównanie użytej pamięci.




## 8.Konfiguracja sprzętowa

Wszystkie testy wykonywałem na komputerze z następującymi podzespołami:

- Procesor: i5 7600k  4x3,8GHz 6MB
- Ram: 16 GB 2400 MHz DDR4