MOWNIT – Laboratorium 5
Iteracyjne metody rozwiązywania układów równań

Mikołaj Wróblewski

1. Pierwszą rzeczą do zrobienia na laboratiorum numer 5 było zaimplementowanie metody Jacobiego, która jest iteracyjną metodą rozwiązywania układów równań. Metodę tę zaimplementowałem i przetestowałem jej działanie na funkcji podobnej do tej używanej przeze mnie w poprzednich zadaniach, jedyną modyfikacją było napisanie generatora macierzy dominantnych diagonalnie, gdyż tylko dla takich macierzy możemy korzystać z metody Jacobiego. Wyniki porównałem z wynikami z funkcji bibliotecznej, z dokładnością do 1e-9. Wynikiem jakiego się spodziewałem, oraz jaki uzyskałem jest następujący komunikat:

```
[mikolaj@mikolaj-pc lab5]$ python jacobi.py
Tests passed
```

Oznacza to iż wszystkie testy się powiodły, a co za tym idzie metoda ta została zaimplementowana poprawnie. Co do metody Jacobiego, trzeba pamiętać że nie jest ona metodą dokładną. Nadal istnieje możliwość, że nasz błąd stanie się większy niż ten z dokładnością do którego porównujemy. Ilość iteracji, które wykonuję wynosi n dla macierzy A o rozmiarze n x n.

Wiedząc, dla jakich macierzy metoda Jacobiego działa, sprawdziłem co stanie się, gdy testy wykonamy dla macierzy niebędących macierzami diagonalnie dominantnymi.

```
[mikolaj@mikolaj-pc lab5]$ python jacobi.py
jacobi.py:47: RuntimeWarning: divide by zero encountered in true_divide
  x = (b - np.dot(R,x)) / D
The matrix test failed
```

Wyniki które otrzymałem, albo zawierają próbę dzielenia przez zero, albo sam komunikat „The matrix test failed” - czyli wyniki rozbiegły się bardziej aniżeli byśmy tego chcieli.

Bardzo ważne jest to, że warunek dominantności diagonalnej macierzy jest tylko warunkiem wystarczającym. Warunkiem koniecznym jest wartość promienia spektralnego macierzy iteracji. Musi ona być mniejsza od 1. Zatem naturalną zmianą w kodzie było dodanie sprawdzania tego warunku.

Przykładem macierzy dominantnej diagonalnie, dla której metoda Jacobiego zawodzi jest macierz:

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

Dla której:

$$D^{-1}R = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix} \quad \Rightarrow \quad \rho(D^{-1}R) = 2\,.$$

Po dodaniu następującej modyfikacji, o której przed chwilą wspomniałem:

```python
# Checking standard convergence condition
iteration_matrix = np.matmul(np.linalg.inv(np.diagflat(D)),R)
spectral = max(np.linalg.eigvals(abs(iteration_matrix)))
if spectral > 1:
    print(spectral)
    print("Convergence condition wasn't met.")
    return x # at this point we will always be returning guess list or list filled with 0
```

Dla tej ciekawej macierzy otrzymujemy:

```
2.0000000000000004
Convergence condition wasn't met.
```

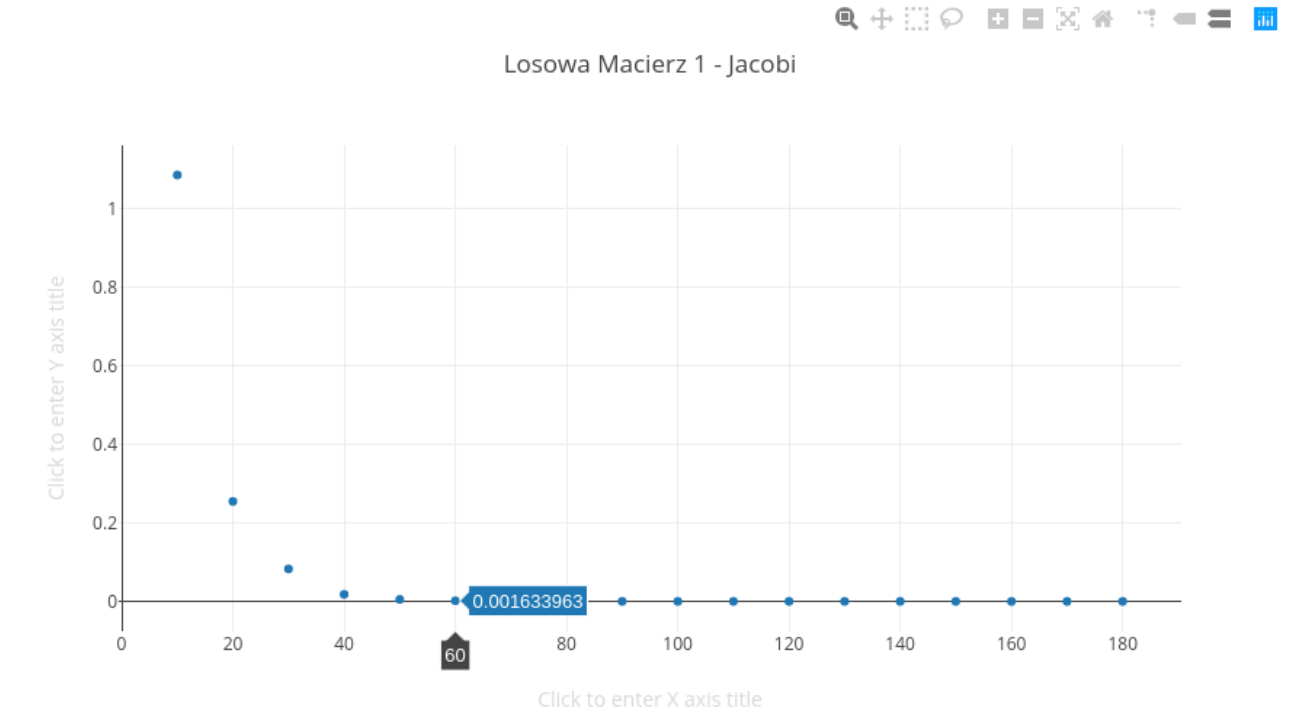
Widzimy, że mimo iż nasza macierz była macierzą dominantną diagonalnie to warunek konieczny nie został spełniony.

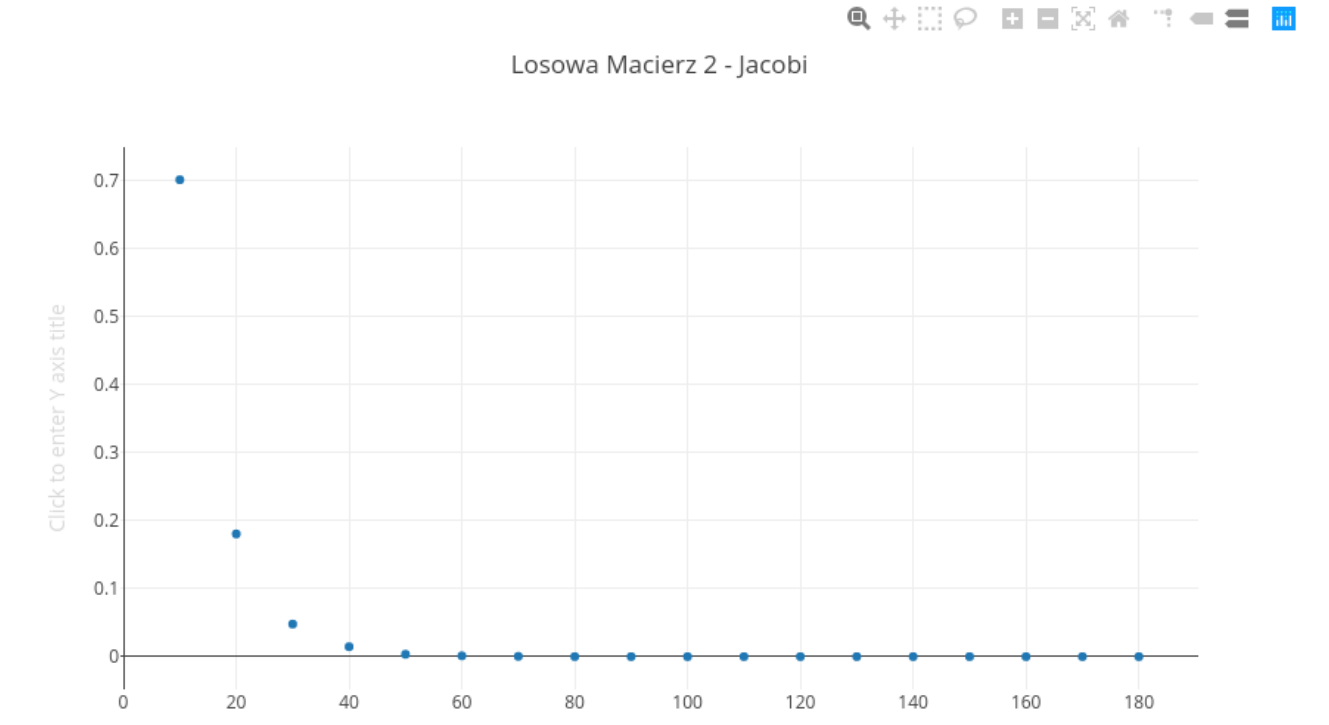
2. Następnie zaimplementowałem metodę Gaussa-Seidela. Dla tego samego generatora macierzy diagonalnie dominantnych przetestowałem jego działanie, otrzymałem to czego należałoby się spodziewać:

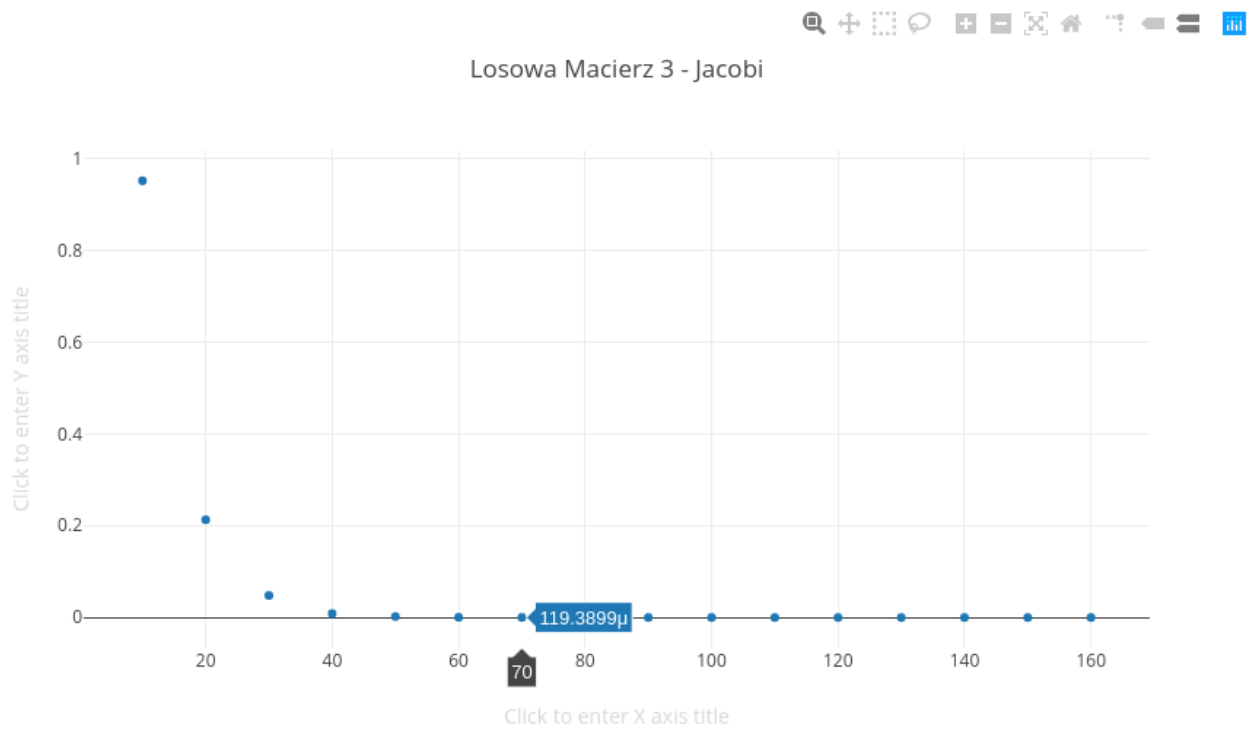
```
[mikolaj@mikolaj-pc lab5]$ python jacobi.py
Jacobi tests:
Tests passed
Gauss tests:
Tests passed
```

Kolejnym krokiem było zapisywanie wartości bezwględnej sumy różnic z kolejnych iteracji wyników, dzieliłem je przez sumę z poprzedniej iteracji. Dzięki temu mogłem wyznaczyć szybkość zbieżności obu implementacji, a następnie je ze sobą porównać na wykresach.

Najpierw wykonałem testy dla macierzy generowanych losowo i różnice umieściłem na wykresie, na osi X umieściłem numer iteracji razy 10 dla lepszego zobrazowania wyników, na osi Y umieściłem różnice pomiędzy kolejnymi wynikami.

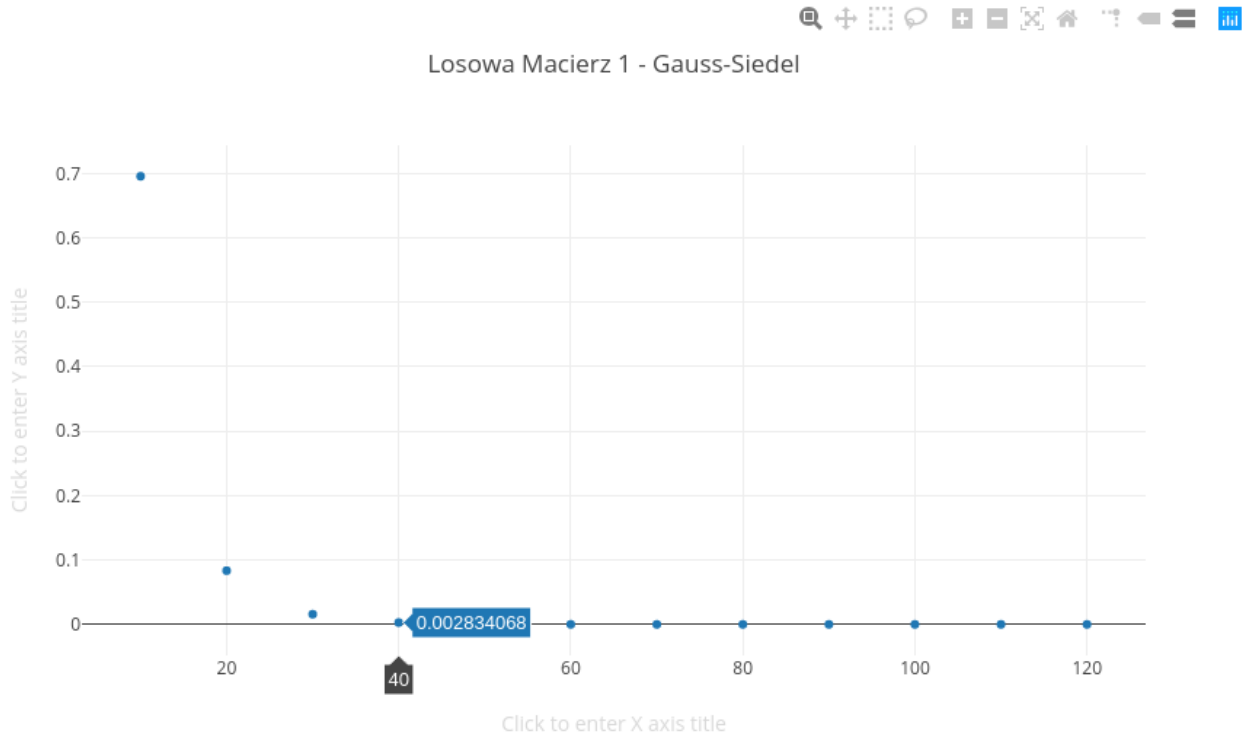

Losowa Macierz 1 - Jacobi
Click to enter Y axis title
1
0.8
0.6
0.4
0.2
0
0.001633963
0
20
40
60
80
100
120
140
160
180
Click to enter X axis title

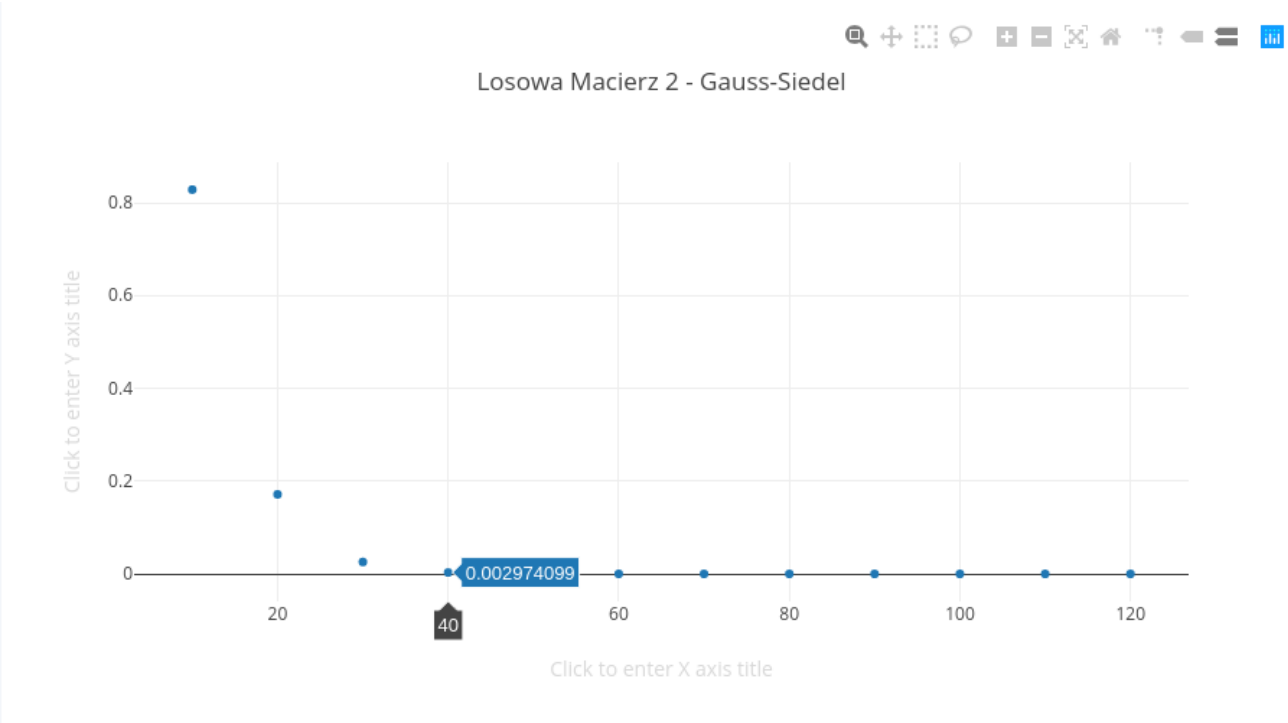

Losowa Macierz 2 - Jacobi
Click to enter Y axis title
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
20
40
60
80
100
120
140
160
180

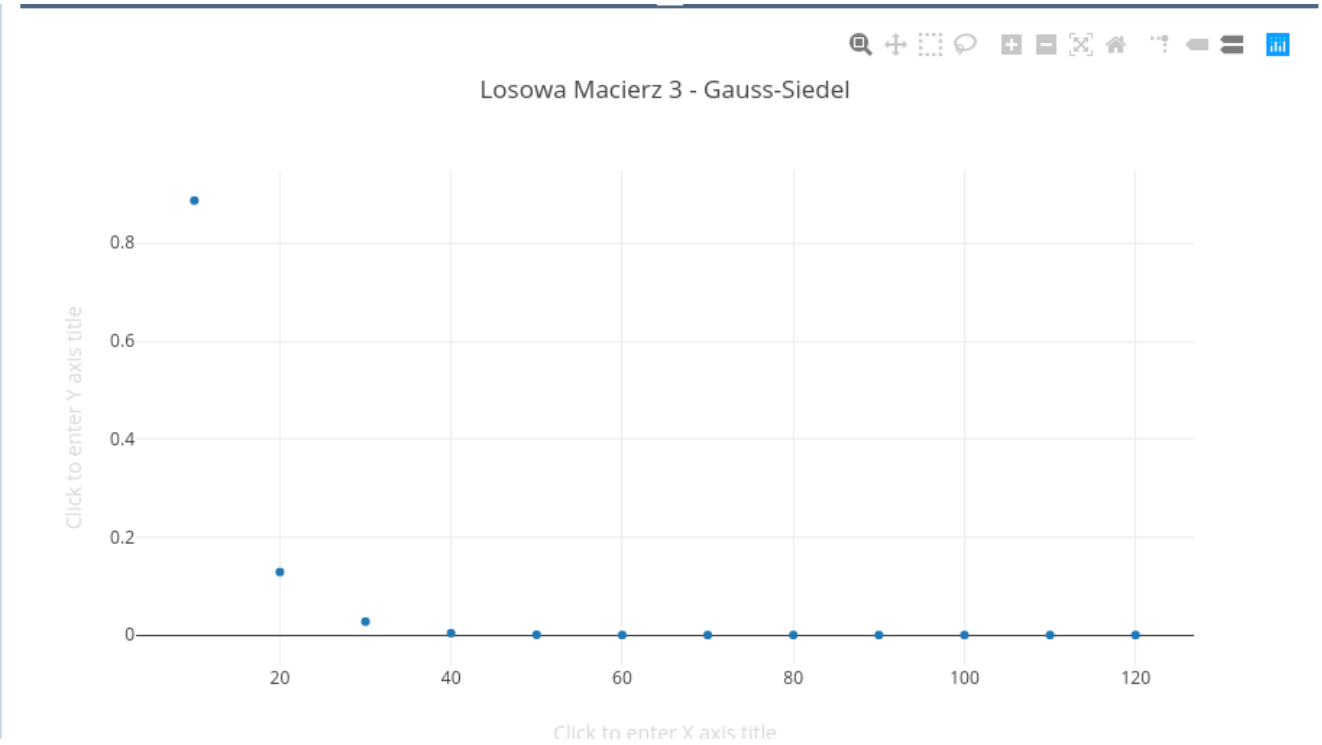


Jak widzimy, dla macierzy całkiem dobrze uwarunkowanych różnice pomiędzy kolejnymi iteracjami zbiegają bardzo szybko do zera.

Teraz wykresy dla metody Gaussa-Siedela:



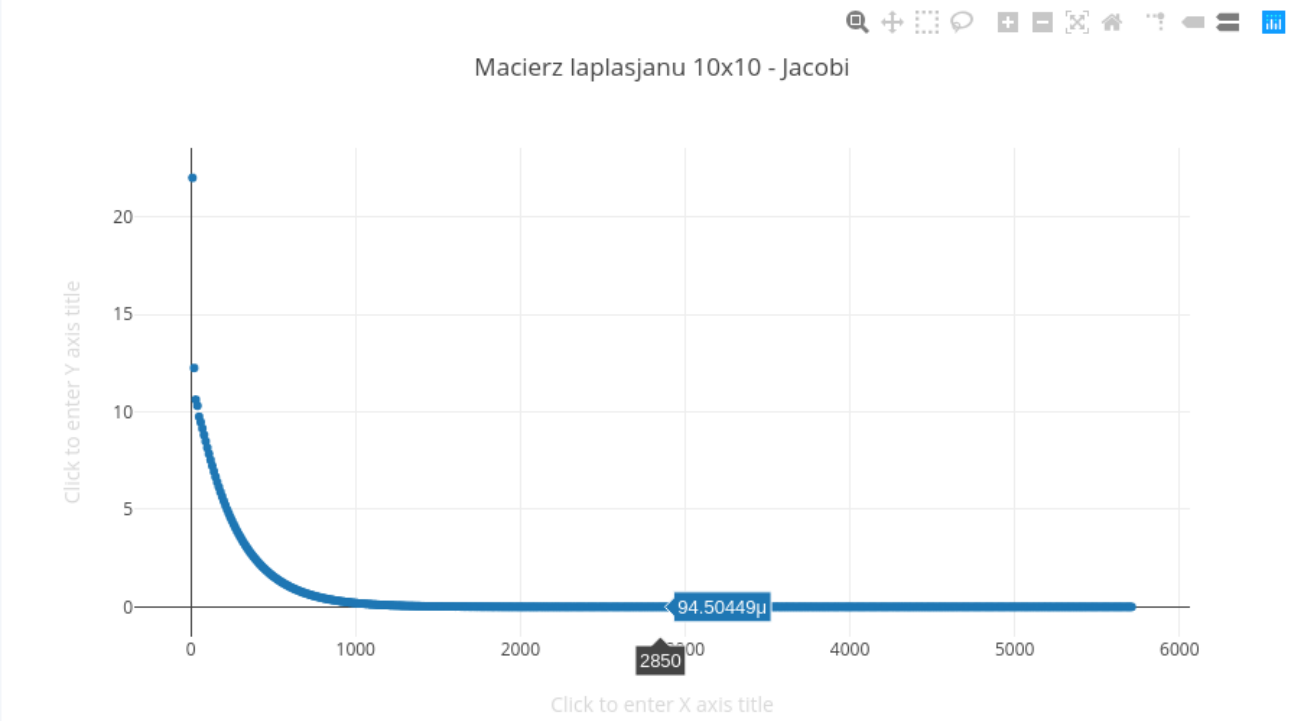




Iteracje u siebie w kodzie kończyłem, kiedy różnica pomiędzy kolejnymi iteracjami była nie większa niż epsilon = 1e-10, ciekawą obserwacją jest to, że dla metody Gaussa-Seidela iteracje były kończone po 12. iteracji.

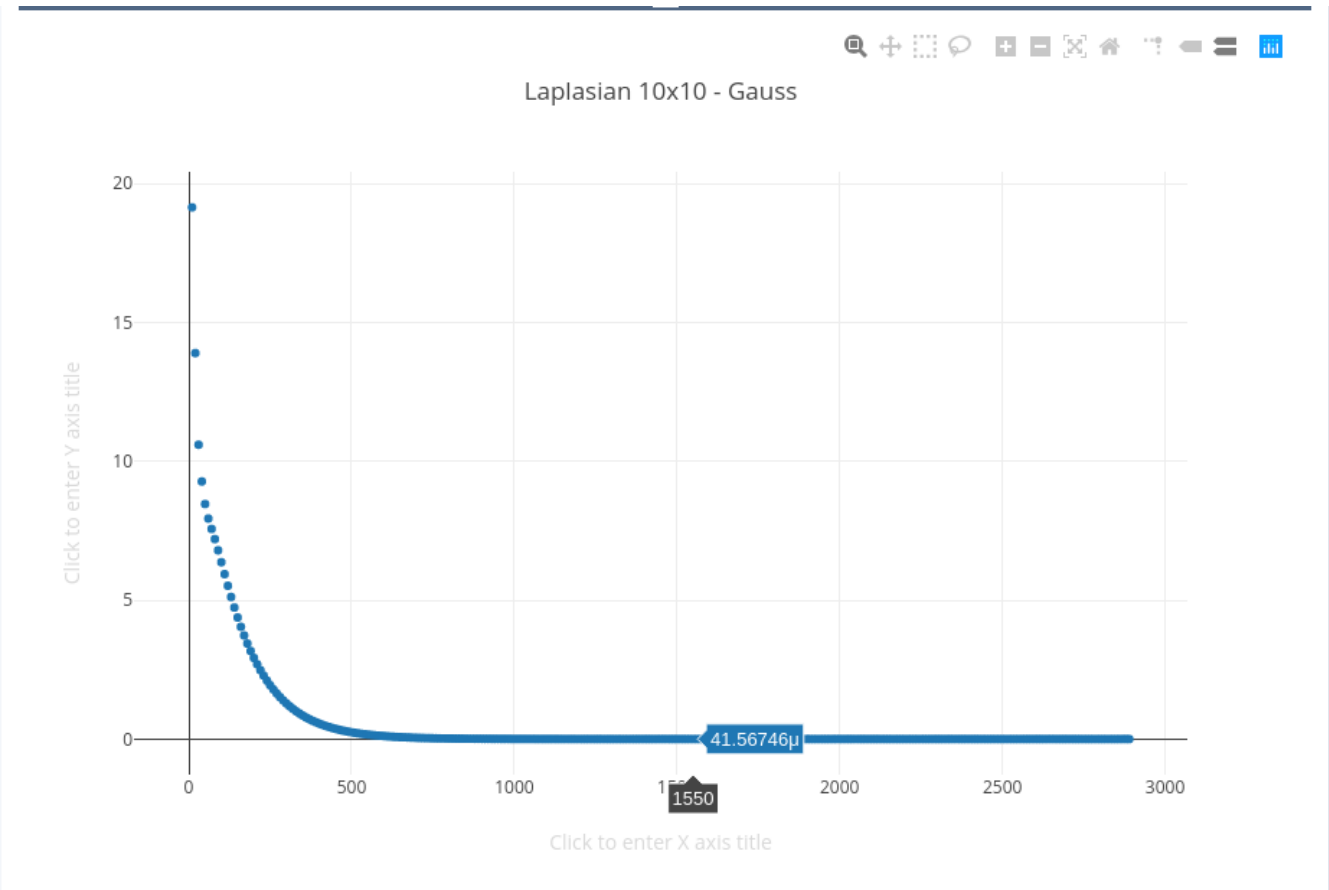
Jak widać dla macierzy dobrze uwarunkowanych różnice również bardzo szybko zbiegały do zera.

Następnie wykonałem test dla jednowymiarowej macierzy Laplasjanu, o rozmiarze 10x10:



Jak widzimy potrzebowaliśmy aż prawie 600 iteracji, by rożnica pomiędzy kolejnymi wynikami była zadowalająca.

Dla tej samej macierzy laplasjanu wykonałem wykres dla metody Gaussa-Seidela.



Jak widzimy liczba iteracji była trochę mniejsza. Co jednak ważne obie próby z użyciem laplasjanu zawiodły i nie dały prawidłowego wyniku.

Reasumując zatem, metody iteracyjne są dobre tylko i wyłącznie dla dobrze uwarunkowanych macierzy, nie są one niestety metodami uniwersalnymi.