MOwNiT laboratorium 2. - Sprawozdanie

Michał Szczurek
Informatya, WIEiT

Grupa poniedziałek 12:50

1 Opis zadania

Zadanie polega na obliczeniu macierzy $(A + uv^T)^{-1}$ dla losowo wygenerowanej macierzy $A \in R^{NxN}$ oraz wektorów $u \in R^{Nx1}$ i $v \in R^{Nx1}$. W tym celu najpierw policzono macierz A^{-1} i $A + uv^T$, a następnie zastosowano trzy algorytmy:

- Algorytm liczący odwrotność $A + uv^T$ korzystający z funkcji numpy.linalg.inv
- Algorytm obliczający wynik przy pomocy wzoru Shermanna-Morrisona

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

- Algorytm obliczający wynik przy pomocy wzoru Shermanna-Morrisona, z dodanym nawiasem

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}(uv^T)A^{-1}}{1 + v^TA^{-1}u}$$

2 Implementacja algorytmów

Poniżej znajduje się kod opisanych algorytmów:

- Algorytmu polegającego na prostym odwróceniu macierzy

```
def simple_inversion(AuvT):
   return np.linalg.inv(AuvT)
```

- Algorytmu Shermanna-Morrisona (1)

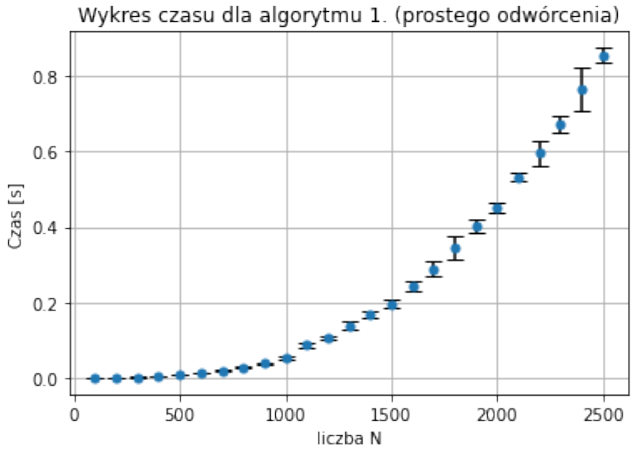
```
def sherman_morrison_v1(u, v, inv_A):
  numerator = (inv_A @ u @ v.T @ inv_A)
  denominator =  (1.0 + v.T @ inv_A @ u)
  res = inv_A -  numerator / denominator
  return res
```

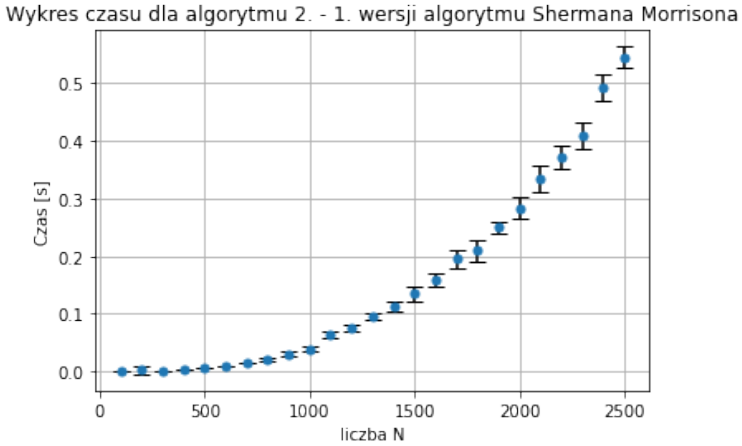
- Algorytmu Shermanna-Morrisona (2)

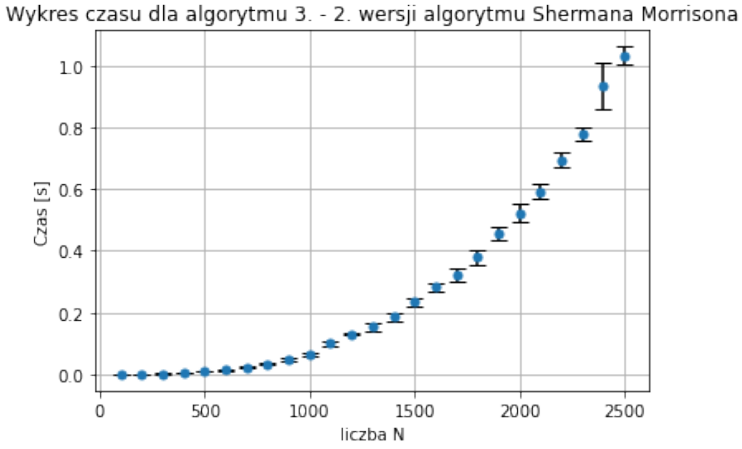
```
def sherman_morrison_v2(u, v, inv_A):
  numerator = (inv_A @ (u @ v.T) @ inv_A)
  denominator =  (1.0 + v.T @ inv_A @ u)
  res = inv_A -  numerator / denominator
  return res
```

3 Porównanie czasów

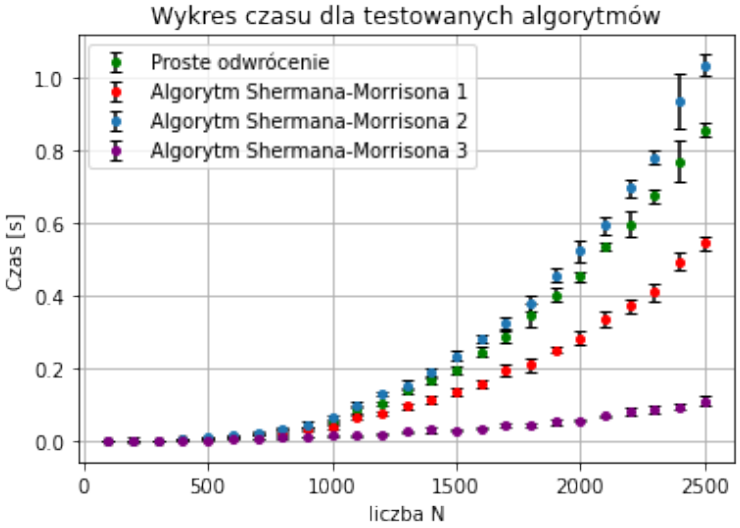
Przetestowano prędkość wykonywania się wszystkich trzech algorytmów dla punktów z zakresu [100, 2500] ze krokiem 100. Wyniki przedstawiono na poniższych wykresach:







Poniżej znajduje się wykres porównujący czasy poszczególnych algorytmów. Wykres uwzględnia algorytm dodatkowy opisany w 5. punkcie sprawozdania Z wykresów wynika, że algorytm 3. był nieznacznie wolniejszy od 1. zdecydowanie najszybszym algorytmem był za to algorytm 2. (około 2 razy szybszy od pozostałych). W następnym punkcie sprawozdania spróbuję podać przyczynę tego stanu rzeczy.



4 Złożoność algorytmów

W tym punkcie spróbuję oszacować złożoność każdego z algorytmów.

- Algorytm 1. - Proste odwrócenie
  Funkcja numpy.linalg.inv, która z kolei wykorzystuje łapack's LU factorization". Jesli macierz nie jest ortogonalna, to sprowadza się to do użycia algorytmu eliminacji Gaussa o złożoności $O(n^3)$.
  Złożoność: $O(n^3)$

- Algorytm 2. - Pierwsza wersja algorytmu Shermanna-Morrisona
  Aby policzyć złożoność algorytmu przeanalizujmy poszczególne jego kroki
  - policzenie w liczniku $A^{-1}u$ - mnożenie macierzy przez wektor ma złożoność $O(n^2)$ i daje wektor.
  - przemnożenie wektora z poprzedniego punktu i v^T ma złożoność $O(n^2)$ i daje macierz.
  - przemnożenie macierzy z poprzedniego punktu i A^{-1} ma złożoność $O(n^3)$ i daje macierz.
  - przemnożenie v^T i A^{-1} w mianowniku ma złożoność $O(n^2)$ i daje wektor.
  - przemnożenie wektora z poprzedniego punktu i u ma złożoność $O(n)$ i daje pojedynczą liczbę.
  - dodanie liczby z poprzedniego punktu i 1 ma złożoność $O(1)$ i daje liczbę.
  - wykonanie dzielenia macierzy przez liczbę ma złożoność $O(n^2)$.

  Po zsumowaniu i uwzględnieniu dwóch operacji transpozycji:
  Złożoność: $O(n^3 + 4n^2 + n + 3)$

- Algorytm 3. - Druga wersja algorytmu Shermanna-Morrisona Aby policzyć złożoność algorytmu przeanalizujmy poszczególne jego kroki
  - przemnożenie u i v^T ma złożoność $O(n^2)$ i daje macierz.
  - przemnożenie A^{-1} macierzy z poprzedniego punktu ma złożoność $O(n^3)$ i daje macierz.
  - przemnożenie macierzy z poprzedniego punktu i A^{-1} ma złożoność $O(n^3)$ i daje macierz.
  - przemnożenie v^T i A^{-1} w mianowniku ma złożoność $O(n^2)$ i daje wektor.
  - przemnożenie wektora z poprzedniego punktu i u ma złożoność $O(n)$ i daje pojedynczą liczbę.
  - dodanie liczby z poprzedniego punktu i 1 ma złożoność $O(1)$ i daje liczbę.
  - wykonanie dzielenia macierzy przez liczbę ma złożoność $O(n^2)$.

Po zsumowaniu i uwzględnieniu dwóch operacji transpozycji:
Złożoność: $O(2n^3 + 3n^2 + n + 3)$
Należy zauważyć, że w algorytmie tym występuje 3 razy więcej operacji o złożoności $O(n^3)$ niż w algorytmie 2., co tłumaczy, dlaczego ta wersja jest około dwa razy wolniejsza(te operacje dominują czas pracy algorytmu).

5 Algorytm dodatkowy

Analizując złożoność algorytmów pomyślałem, że może dobrym pomysłem byłoby zmienić nawiasowanie w następujący sposób:

$$(A + uv^T)^{-1} = A^{-1} - \frac{(A^{-1}u)(v^T A^{-1})}{1 + v^T A^{-1} u}$$

Jako że mnożenie macierzy przez wektor/ wektora transponowanego przez macierz ma złożoność $O(n^2)$ i daje wektor, a mnożenie dwóch wektorów ma również złożoność $O(n^2)$ to cały algorytm powinien mieć złożoność $O(n^2)$. Wykonałem testy i wykres dla tego algorytmu (używając jednak innych, równoznacznych funkcji, tak by nie naruszać gotowego już kodu dla poprzedniej części zadania). Algorytm okazał się być istotnie szybszy.