Algorytmy Geometryczne
Sprawozdanie 2

Aga Patro
czw_13.30_A

1. Specyfikacja sprzętu i narzędzia wykorzystane w realizacji

System: Debian Linux Parrot OS x64
Procesor: AMD Ryzen 5 4500U, 6 rdzeni, 6 wątków, 4.00GHz
Pamięć RAM: 16 GB
Środowisko: Jupyter Notebook
Język: Python 3
Narzędzie pomocnicze: plik *geometria.ipynb* dostarczony z poleceniem, biblioteki *numPy*, *math, random*, *matplotlib* oraz *time*

2. Temat ćwiczenia

Celem ćwiczenia była implementacja i przetestowanie algorytmów obliczających otoczkę wypukłą, czyli uwypuklenie podzbioru przestrzeni liniowej – najmniejszy zbiór wypukły zawierający ten podzbiór.

3. Wprowadzenie teoretyczne

By obliczyć otoczkę wypukłą, korzystam z dwóch algorytmów: Grahama i Jarvisa

3.1 Algorytm Grahama

Algorytm ten ma złożoność O(nlogn), gdzie n to liczba punktów w zbiorze. Przebiega w następujący sposób:

1. Wybieramy punkt p_0 czyli punkt o najmniejszej współrzędnej y. Jeśli mamy punkty o tej samej współrzędnej y to kierujemy się współrzędną x.
2. Sortujemy punkty względem kąta α pomiędzy wektorem $(p_0 p_i)$, a dodatnią osią OX układu współrzędnych.
3. Usuwamy punkty tworzące taki sam kąt α z wyjątkiem punktu oddalonego najbardziej od p_0.
4. Odkładamy punkty p_0, p_1, p_2 na stosie.
5. Przeglądamy posortowaną listę punktów i zaczynając od p_3:
   - Ściągamy wierzchołki ze stosu tak długo jak punkt p_i nie jest położony na lewo od prostej $(p_{t-1} p_t)$, gdzie t to indeks stosu
   - Dodajemy p_i na szczyt stosu
6. Zwracamy stos jako wynik.

3.2 Algorytm Jarvisa

Algorytm ma złożoność O($n * k$), gdzie k to ilość punktów należących do otoczki wypukłej, a n jest liczbą wszystkich punktów należących do zbioru.
Przebiega w następujący sposób:

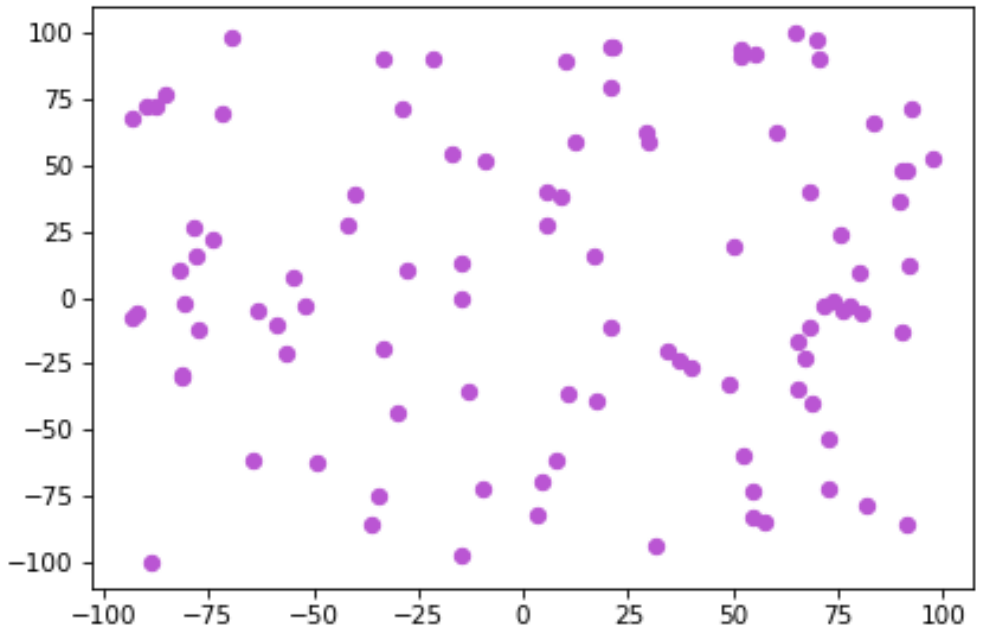
1. Wybieramy punkt $p_0$czyli punkt o najmniejszej współrzędnej y. Jeśli mamy punkty o tej samej współrzędnej y to kierujemy się współrzędną x.
2. Znajdujemy punkt N dla którego kąt liczony przeciwnie do ruchu wskazówek zegara w odniesieniu do ostatniej krawędzi otoczki jest najmniejszy i dodajemy go do listy punktów otoczki wypukłej.
3. Powtarzamy punkt 2. dopóki N != p_0.

4. Realizacja

4.1 Generowanie i ilustrowanie punktów

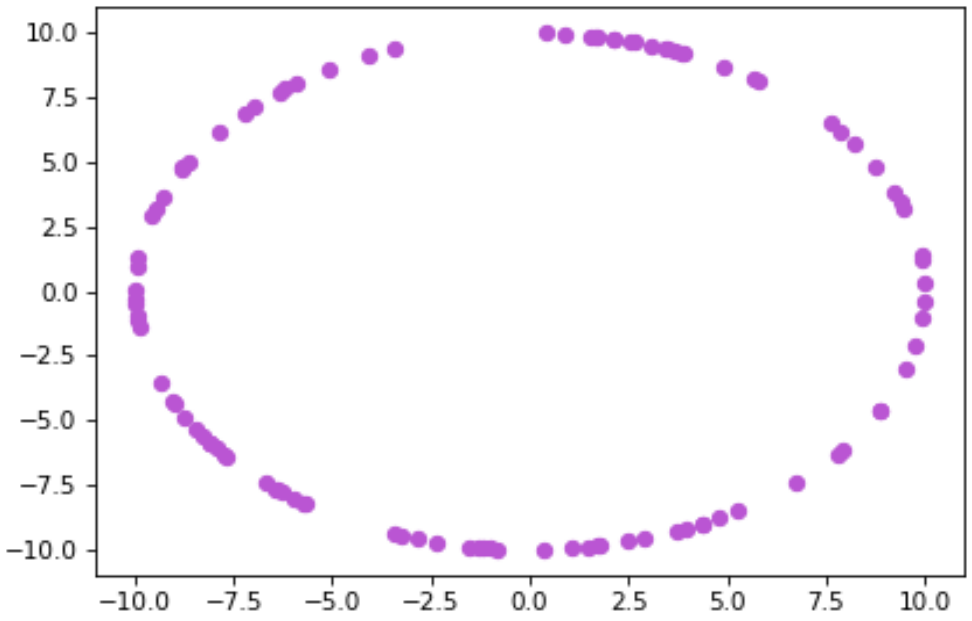
By wygenerować punkty użyłam funkcji *randint()* i *uniform()* z biblioteki *random*. Funkcja ta zwraca losową liczbę z przedziału podanego jako argument.
Dodatkowo, w podpunkcie 2 skorzystałam z funkcji *cos()* i *sin()* z biblioteki *math.*
Ilustracje wszystkich zbiorów zostały zrealizowane za pomocą funkcji z pliku *geometria.ipynb* opartego o bibliotekę matplotlib

1) Zbiór 1 - 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100]



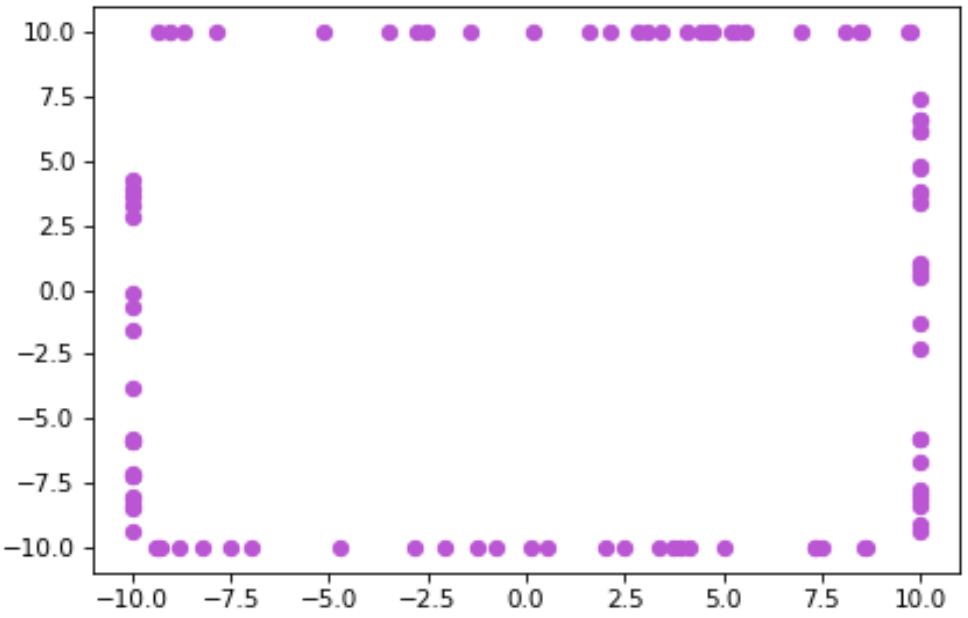
Wykres 4.1.1 Zbiór 1

2) Zbiór 2 - zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10



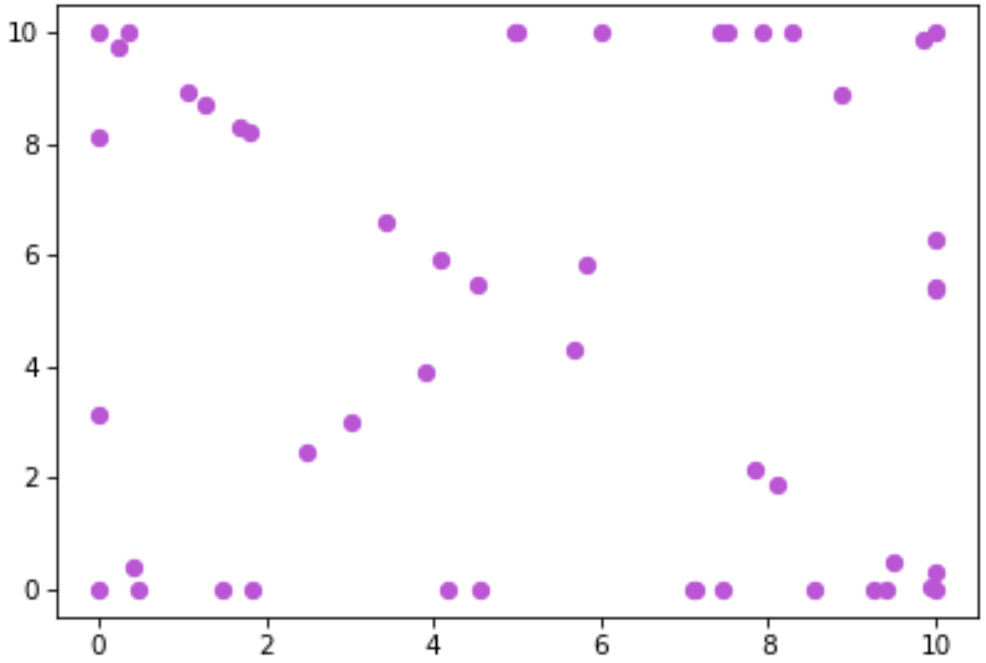
Wykres 4.1.2 Zbiór 2

3) Zbiór 3 - 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10),



Wykres 4.1.3 Zbiór 3

4) Zbiór 4 -wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo w sposób następujący: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu.



Wykres 4.1.4 Zbiór 4

4.2 Wyznaczanie otoczki wypukłej

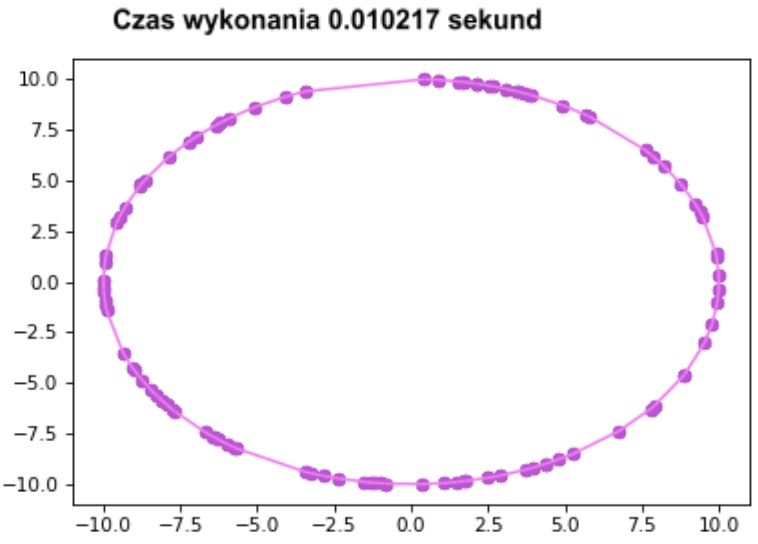
By wyznaczyć otoczkę wypukłą zaimplementowałam algorytmy Grahama i Jarvisa. Dla obu algorytmów zdecydowałam się zastosować własną metodę liczenia wyznacznika 3x3 oraz wartość e = 10^{-10}. W implementacji dodałam zmienną "*want_to_see_chart*" która określa, czy chcemy by algorytm generował wizualizację. Dla algorytmów bez wizualizacji zmierzyłam czas wykonania i porównałam je ze sobą w tabelach.

By zmierzyć czas wykonywania, zaimplementowałam funkcję *measure_time*, w której skorzystałam z biblioteki *time*.

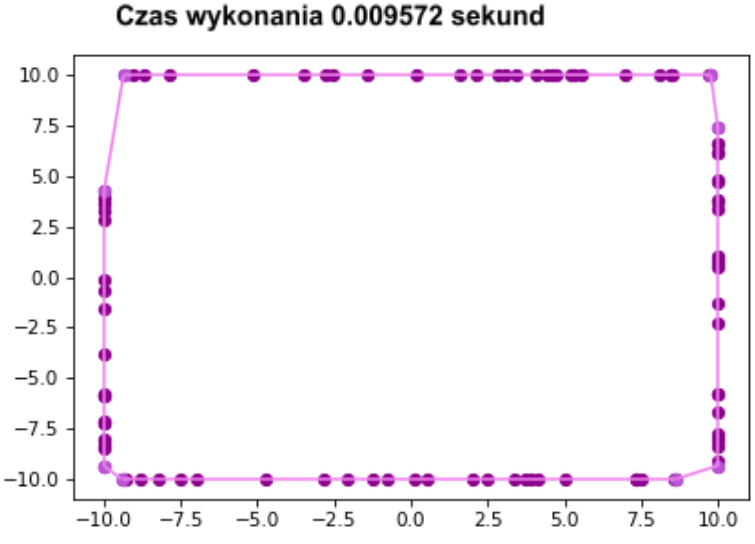
Dodatkowo zaimplementowałam funkcję *save_records*, która zapisuje konkretne wyniki do pliku o rozszerzeniu *.txt*.

4.2.1 Wizualizacja otoczki wypukłej dla podstawowych zbiorów - algorytm Grahama

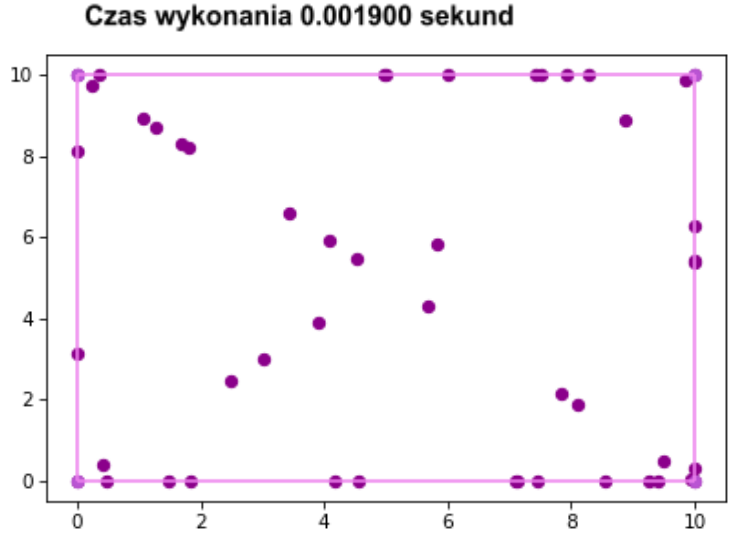
Wykres 4.2.1.1 Otoczka wypukła dla zbioru 1



Wykres 4.2.1.2 Otoczka wypukła dla zbioru 2

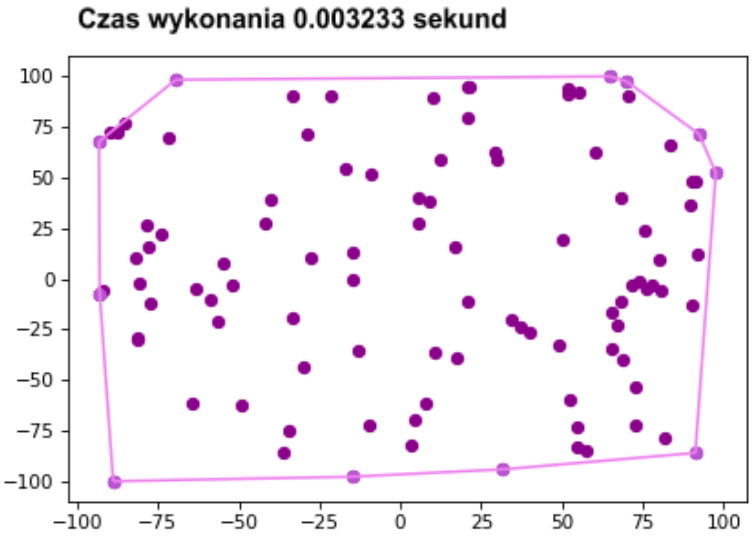


Wykres 4.2.1.3 Otoczka wypukła dla zbioru 3

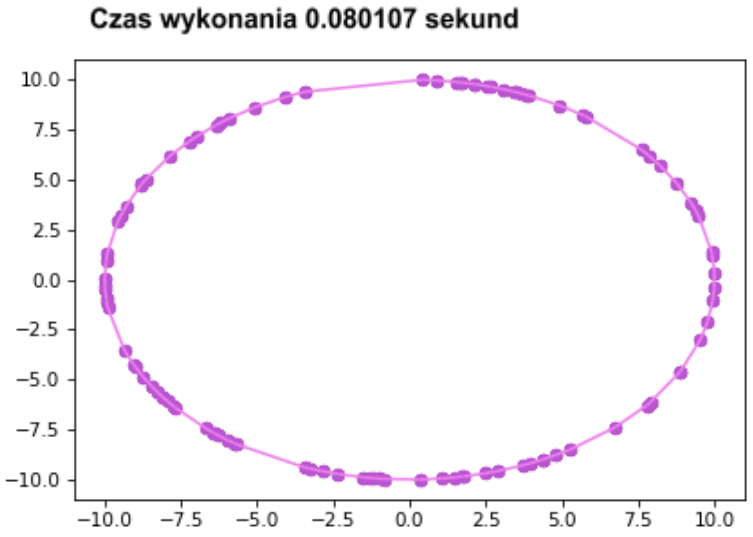


Wykres 4.2.1.4 Otoczka wypukła dla zbioru 4

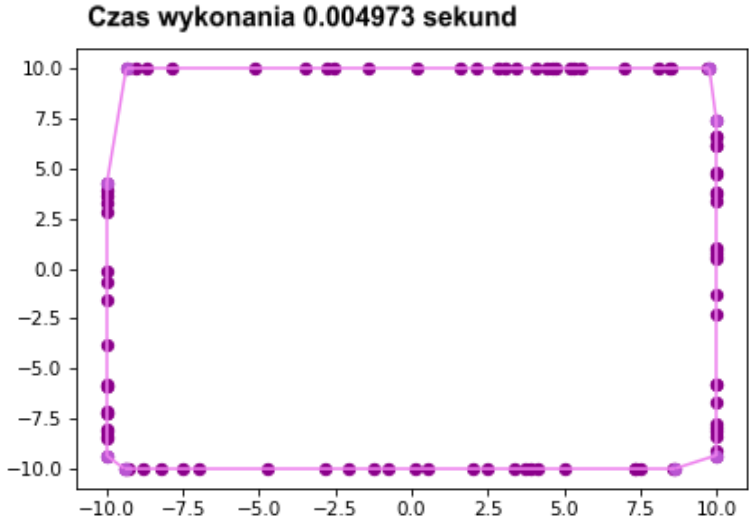
4.2.2 Wizualizacja otoczki wypukłej dla podstawowych zbiorów - algorytm Jarvisa



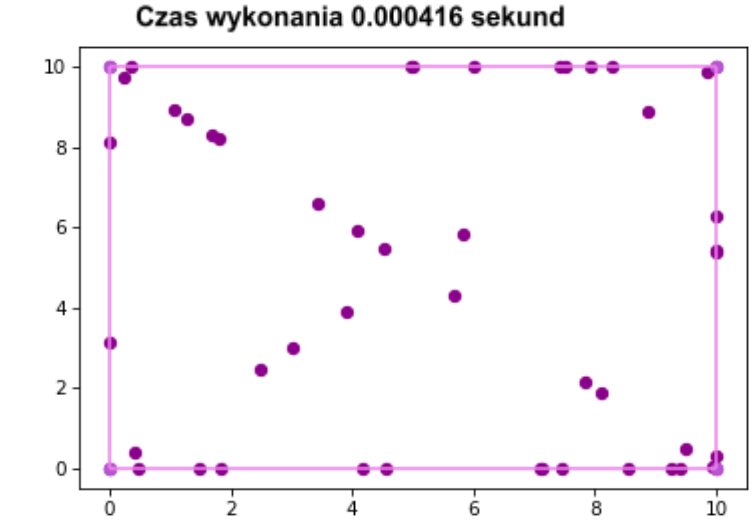
Wykres 4.2.2.1 Otoczka wypukła dla zbioru 1



Wykres 4.2.2.2 Otoczka wypukła dla zbioru 2



Wykres 4.2.2.3 Otoczka wypukła dla zbioru 3



Wykres 4.2.2.4 Otoczka wypukła dla zbioru 4

Dla podstawowych zbiorów ukazanych powyżej można zauważyć, że algorytm Jarvisa jest szybszy dla danej ilości punktów oraz, że poprawność algorytmów jest taka sama.

4.2.3 Przedstawienie wyników pomiaru czasu

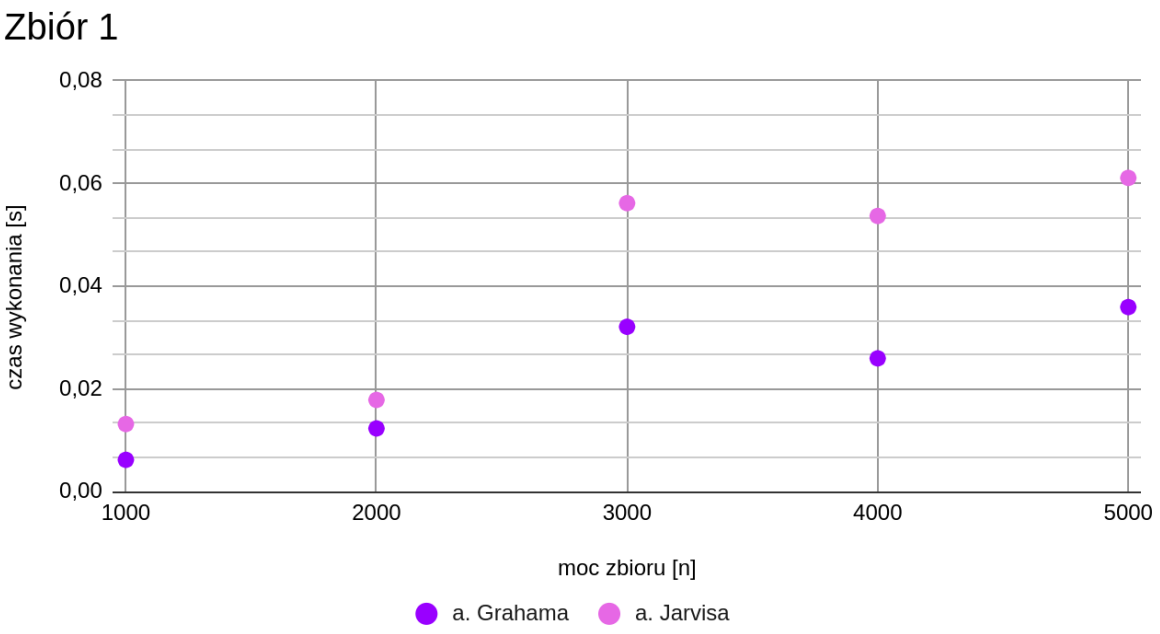
W poniższych tabelach i wykresach przedstawiłam wyniki pomiarów czasu dla różnej liczby punktów, gdzie:

- n oznacza ilość punktów w zbiorze,
- p_g, p_j oznaczają ilość punktów należących do otoczki wypukłej (g dla Grahama, j dla Jarvisa),
- t_g oznacza czas dla algorytmu Grahama, czas mierzony w sekundach
- t_j oznacza czas dla algorytmu Jarvisa, czas mierzony w sekundach

4.2.3.1 Wyniki dla zbioru 1

n	p_g	p_j	t_g	t_j
1000	23	23	0.006188	0.013161
2000	17	17	0.012304	0.017857
3000	22	22	0.032097	0.056174
4000	27	27	0.025950	0.053678
5000	24	24	0.035935	0.061089

Tabela 4.2.3.1 Wyniki pomiarów czasu dla obu algorytmów, zbiór 1



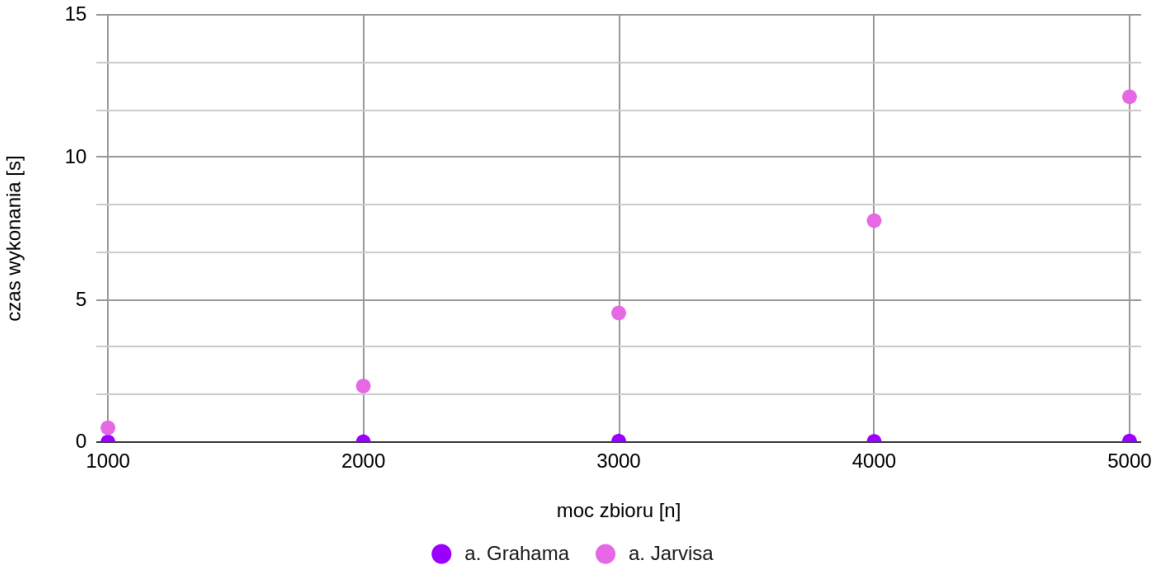
Wykres 4.2.3.1 Wyniki pomiarów czasu dla obu algorytmów, zbiór 1

4.2.3.2 Wyniki dla zbioru 2

n	p_g	p_j	t_g	t_j
1000	999	999	0.005136	0.501747
2000	1992	1992	0.011463	1.968332
3000	2979	2979	0.033025	4.533003
4000	3955	3955	0.024034	7.774012
5000	4894	4895	0.033694	12.12292

Tabela 4.2.3.2 Wyniki pomiarów czasu dla obu algorytmów, zbiór 2

Zbiór 2



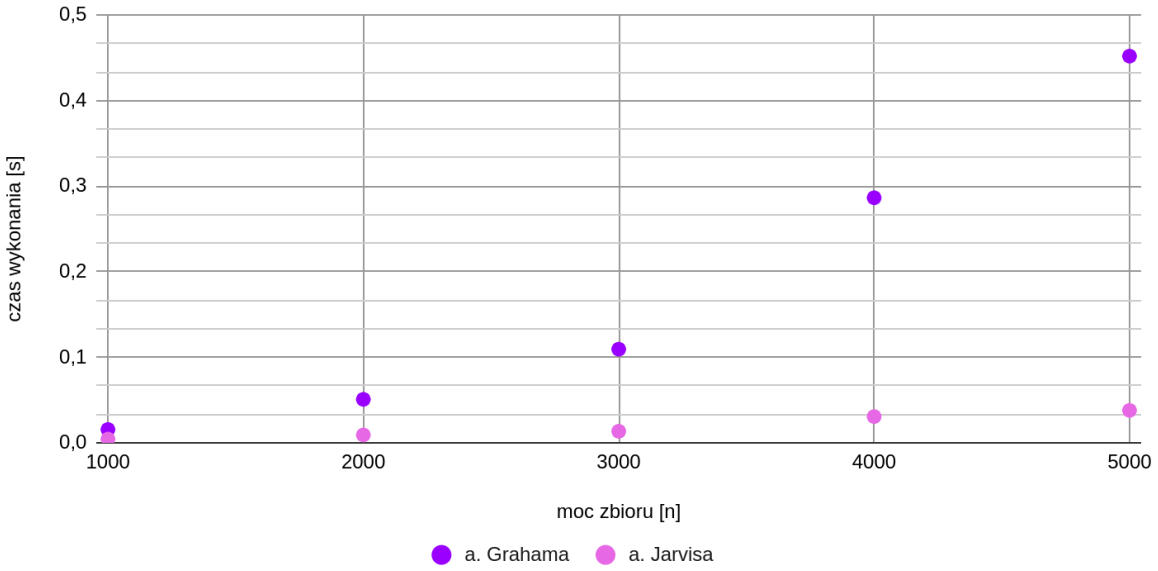
Wykres 4.2.3.2 Wyniki pomiarów czasu dla obu algorytmów, zbiór 2

4.2.3.3 Wyniki dla zbioru 3

n	p_g	p_j	t_g	t_j
1000	8	8	0.015622	0.004437
2000	8	8	0.050852	0.009230
3000	8	8	0.109448	0.013549
4000	8	8	0.286390	0.030715
5000	8	8	0.451883	0.037955

Tabela 4.2.3.3 Wyniki pomiarów czasu dla obu algorytmów, zbiór 3

Zbiór 3



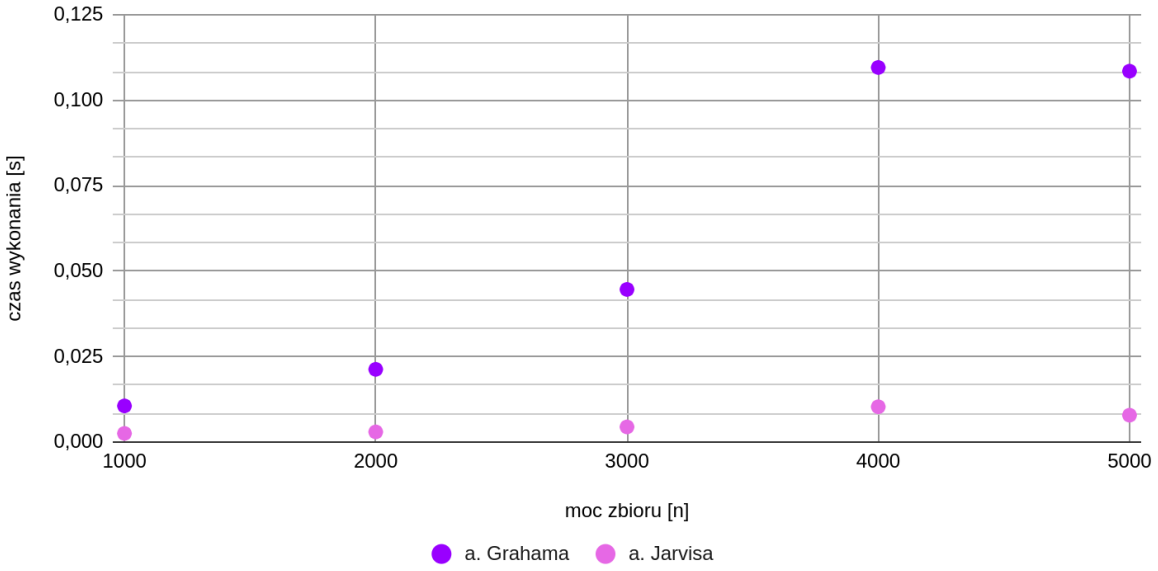
Wykres 4.2.3.3 Wyniki pomiarów czasu dla obu algorytmów, zbiór 3

4.2.3.4 Wyniki dla zbioru 4

n	p_g	p_j	t_g	t_j
1000	4	4	0.010592	0.002539
2000	4	4	0.021283	0.002979
3000	4	4	0.044656	0.004435
4000	4	4	0.109595	0.010335
5000	4	4	0.108515	0.007874

Tabela 4.2.3.4 Wyniki pomiarów czasu dla obu algorytmów, zbiór 4

Zbiór 4



Wykres 4.2.3.4 Wyniki pomiarów czasu dla obu algorytmów, zbiór 4

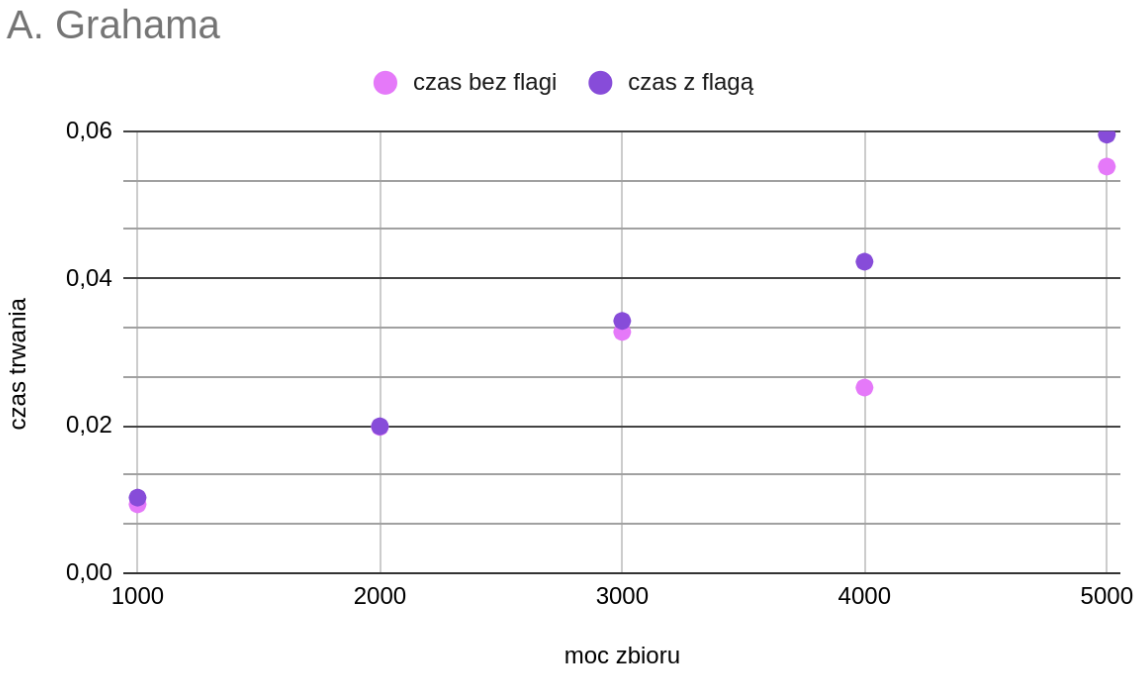
4.2.4 Przedstawienie wyników pomiaru czasu bez flagi w funkcjach

Przeprowadziłam pomiary dla nowego zbioru 1, przedstawiłam wyniki w tabeli poniżej, oddzielnie dla algorytmu Grahama (t_g) i oddzielnie dla algorytmu Jarvisa (t_j). Czas był mierzony w sekundach.

4.2.4.1 Wyniki dla algorytmu Grahama

n	$t_{g\ z\ flagą}$	$t_{g\ bez\ flagi}$
1000	0.0102519	0.0093395
2000	0.0199298	0.0198450
3000	0.0342335	0.0327370
4000	0.0422701	0.0251884
5000	0.0595118	0.0551805

Tabela 4.2.1.1 Wyniki pomiarów czasu algorytmu Grahama

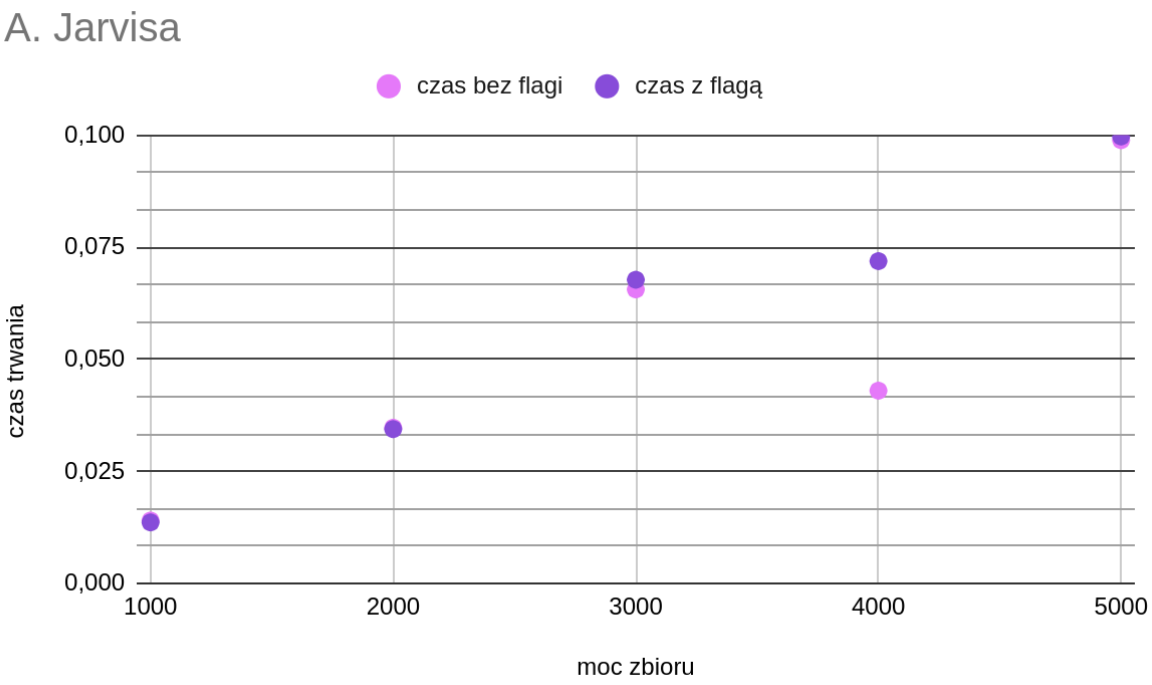


Wykres 4.2.4.1 Wyniki pomiarów czasu dla algorytmu Grahama

4.2.4.2 Wyniki dla algorytmu Jarvisa

n	$t_{j\ z\ flagą}$	$t_{j\ bez\ flagi}$
1000	0.0136024	0.0140206
2000	0.0344126	0.0347301
3000	0.0677475	0.0655996
4000	0.0719032	0.0429975
5000	0.0997195	0.0988914

Tabela 4.2.1.2 Wyniki pomiarów czasu algorytmu Jarvisa



Wykres 4.2.4.2 Wyniki pomiarów czasu algorytmu Jarvisa

Jak widać powyżej, nie ma znaczącej różnicy pomiaru czasu dla funkcji z flagą lub bez flagi. Wyjątkiem jest tutaj zbiór o mocy 4000, gdzie czas bez flagi był znacznie niższy.

5. Podsumowanie

5.1 Obserwacje

Dla mniejszej dokładności błędu ($e = 10^{-12}$ i mniejsze), dla zbiorów 3 i 4 algorytm Jarvisa bardzo długo nie zwracał wyniku i powodował przegrzewanie sprzętu. Dlatego przyjęłam $e = 10^{-10}$. Pomimo długich zastanowień, nie udało mi się ustalić co było tego przyczyną.

Dla zbioru 2 zmierzyłam ilość punktów dla obu algorytmów przyjmując e=10^{-8}. Różnica punktów znowu się pojawiła. Dla e = 10^{-6} różnica w klasyfikacji była jeszcze większa. Dopiero dla e = 10^{-11} algorytmy zwróciły taką samą liczbę punktów.

5.2 Wnioski

1) Dla zbiorów 1, 3, 4 nie zaobserwowałam różnicy w poprawności algorytmów. Oba algorytmy działały poprawnie dla tolerancji $e = 10^{-10}$. Natomiast dla zbioru 2 o $n = 5000$, zauważyłam różnicę w klasyfikacji punktów do otoczki wypukłej o 1 punkt. Możliwe że jest to spowodowane błędną klasyfikacją punktu wyznacznikiem.

2) Dla zbioru 1 czas dla obu algorytmów zmieniał się podobnie (można zauważyć podobny wzrost i spadek, np dla $n = 3000$ i $n = 4000$), jednakże algorytm Grahama okazał się szybszy. Myślę, że zostało to spowodowane tym, że funkcja $log(n)$ rosła wolniej niż wartość k.

3) Dla zbioru 2 algorytm Grahama miał podobny czas działania dla każdej ilości punktów. Natomiast algorytm Jarvisa miał złożoność $O(n^2)$, gdyż wszystkie punkty należały do otoczki wypukłej. Dlatego okazał się o wiele wolniejszy od algorytmu Grahama.

4) Dla zbioru 3 czas wykonywania obu algorytmów był zgodny z przewidywaną złożonością czasową, tj $O(nlogn)$ dla algorytmu Grahama i $O(k * n)$ dla algorytmu Jarvisa.

5) Algorytm Jarvisa okazał się lepszy dla zestawów, gdzie mało punktów wchodziło w skład otoczki wypukłej.

6) Zbiór 1 został zaproponowany by pokazać działanie algorytmu dla przypadkowych danych. Zbiór 2 został zaproponowany by pokazać pesymistyczną złożoność algorytmu Jarvisa. Zbiory 3 i 4 zostały zaproponowane by pokazać przewagę algorytmu Jarvisa nad algorytmem Grahama.