Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Algorytmy geometryczne

Ćwiczenie 2

Autor:
Andrzej Zaborniak
Informatyka, rok II gr. 4

10.11.2022 r.

1 Specyfikacja techniczna

Parametry techniczne komputera na którym zostało wykonane ćwiczenie:
Procesor: Intel® Core™ i7-5600U CPU @ 2.60GHz × 4
Karta graficzna: Mesa Intel® HD Graphics 5500 (BDW GT2)
Pamięć RAM: 8,0 GB
System operacyjny: Ubuntu 22.04.1 LTS
Wersja GNOME: 42.4
Użyty język programowania: Python 3.10.6
Wykorzystany program: Jupyter Notebook

1.1 Narzędzie graficzne

Wykresy wizualizujące rozmieszczenie punktów na płaszczyźnie jak i kolejne kroki działania zaimplementowanych algorytmów zostały wykonane przy użyciu funkcji zawartych w bibliotekach Seaborn oraz Pandas zaś przedsatwienie graficzne działania algorytmów: Graha oraz Jarvisa zostało stworzone z pomocą użycia biblioteki matplotlib.pyplot.

2 Cel ćwiczenia

Głównym zadaniem które należało wykonać na laboratorium drugie, była implementacja podstawowych algorytmów do wyznaczania otoczki wypukłej na zadanych zbiorach punktów wyznaczonych na płaszczyźnie. Porównanie ich skuteczności działania oraz czasu w jakim dane algorytmy się wykonują.
Definicja otoczki wypukłej:
Otoczka wypukła na zbiorze S - najmniejszy wielokąt wypukły zawierający S.

3 Algorytmy które należało zaimplementować

3.1 Algorytm Grahama

3.1.1 Opis działania algorytmu

Algorytm Grahama składa się z czterech części:

1. W zbiorze S wybieramy punkt p_0 o najmniejszej współrzędnej y. Jeżeli jest kilka takich punktów, to wybieramy ten z nich, który ma najmniejszą współrzędną x.
2. Sortujemy pozostałe punkty ze względu na kąt, jaki tworzy wektor (p_0,p) z dodatnim kierunkiem osi x. Jeśli kilka punktów tworzy ten sam kąt, usuwamy wszystkie z wyjątkiem najbardziej oddalonego od p_0 . Uzyskanym ciągiem jest $p_0,\ p_1,\ p_2, ..., p_m$.
3. Do początkowego stosu wkładamy punkty p_0, p_1, p_2.

4. while i < m do
   if p_i leży na lewo od prostej (p_{t-1} , p_t))
   then
   push(p_i), i ← i+1
   else
   pop(s)

Do wyznaczenia kąta kąta pomiędzy puntakmi zostałą wykorzystana funkcja *orient*:

$$b < c \Leftrightarrow \text{orient(a,b,c)} > 0$$
$$b = c \Leftrightarrow \text{orient(a,b,c)} = 0$$
$$b > c \Leftrightarrow \text{orient(a,b,c)} < 0$$

3.1.2 Złożoność obliczeniowa

W algorytmie zostaje wykorzystane sortowanie, co sprawia, że koszt nie może zejść poniżej czasu O(nlogn), szuaknie minimum zajmuje czas O(n), krok 4 czas O(n-3), co w ostateczności daje czas O(nlogn).

3.2 Algorytm Jarvisa

3.2.1 Opis działania algorytmu

Algorytm Jarvisa jest również nazywany algorytmem owijania prezentu, a jego kroki prezentują się następująco:

znajdź punkt i_0 z S o najmniejszej współrzędnej y; $i \leftarrow i_0$
repeat
    for j ≠ i do
        znajdź punkt, dla którego kąt liczony przeciwnie do
        wskazówek zegara w odniesieniu do ostatniej krawędzi otoczki jest najmniejszy
    niech k będzie indeksem punktu z najmniejszym kątem
    zwróć (p i, p k) jako krawędź otoczki
    i ← k
until $i = i_0$

3.2.2 Złożoność obliczeniowa

Złożoność jest rzędu O(n^2).

4 Generacja punktów na płaszczyźnie

Zbiory punktów które należało wygenerować, a następnie uruchomić wizualizację graficzną:

a) zbiór zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100],

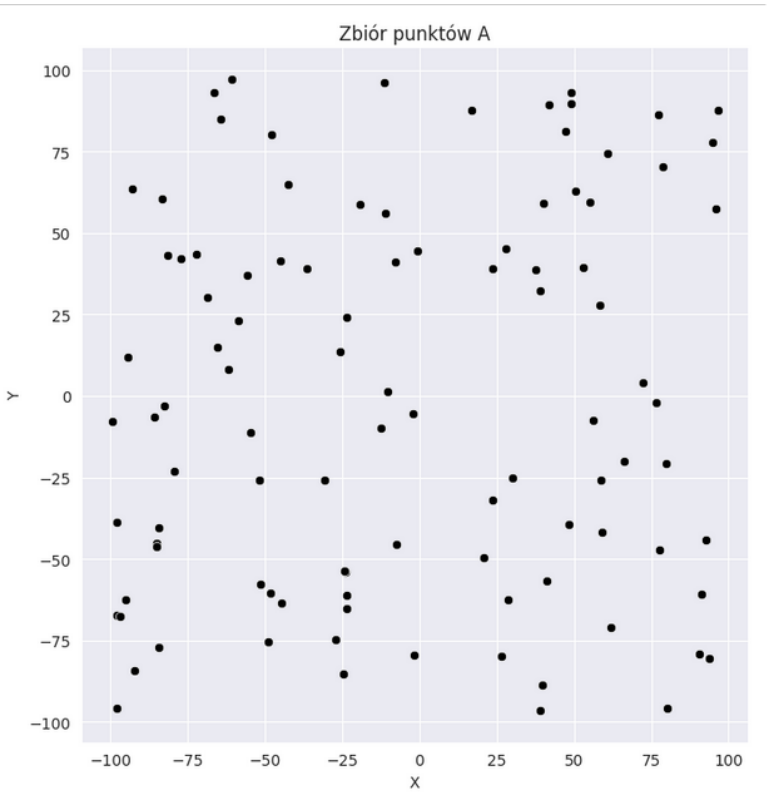
b) zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10,

c) zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10),

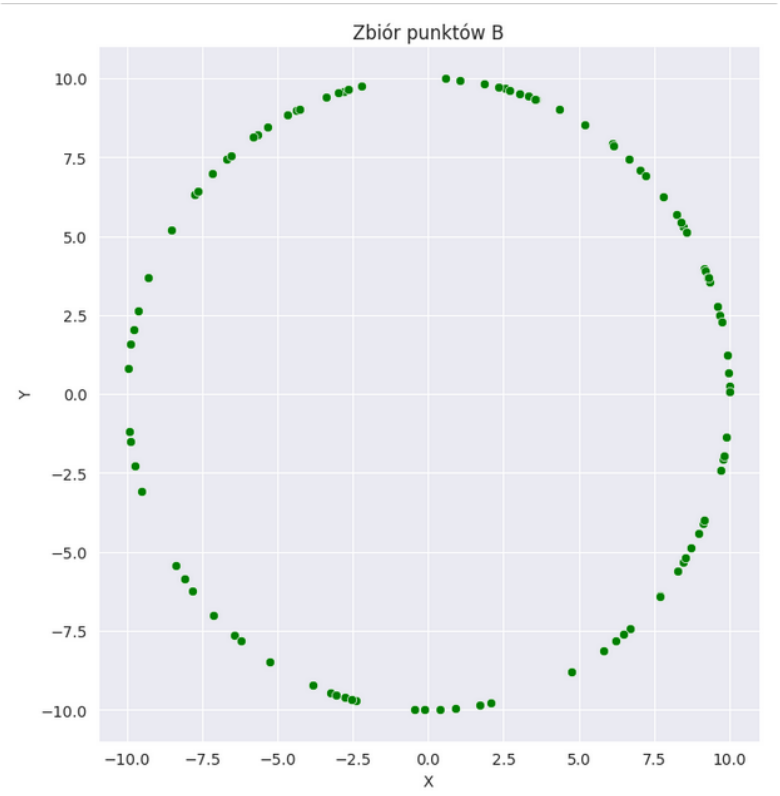
d) zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz punkty wygenerowane losowo w sposób następujący: po 25 punktów na dwóch bokach kwadratu leżących na osiach i po 20 punktów na przekątnych kwadratu.

Do wygenerowania wyżej wymienionych zbiorów punktów użyto funkcji uniform z biblioteki numpy, która znajduje się w module random. Użycie tej funkcji jest znacznie efektywniejsze w porównaniu do standardowej zawartej w bibliotece Python, ponieważ ma szybszy czas działania oraz zapewnia większą liczbę rozkładów prawdopodobieństwa do wyboru.

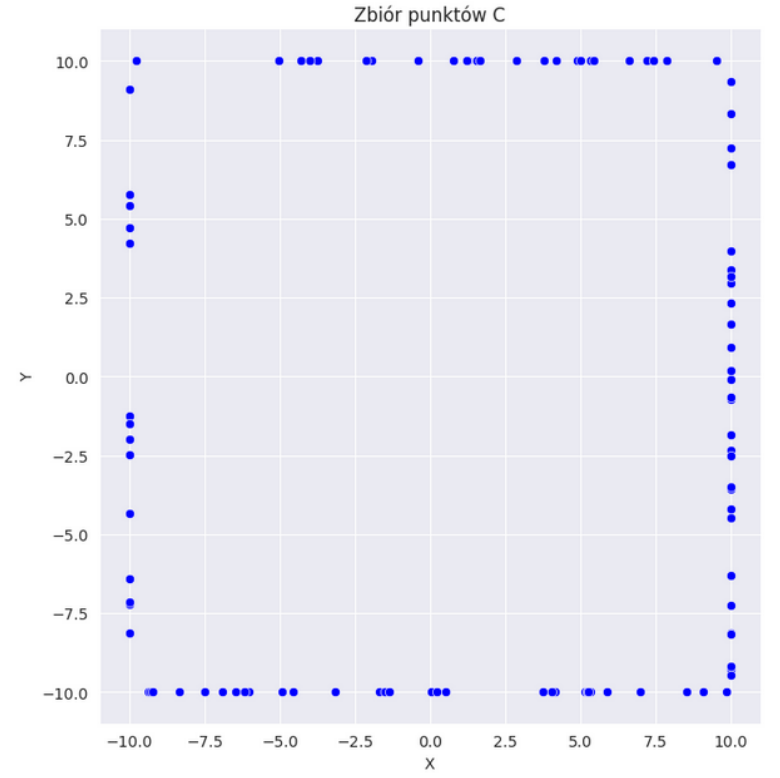
4.1 Wizualizacja graficzna zbiorów:



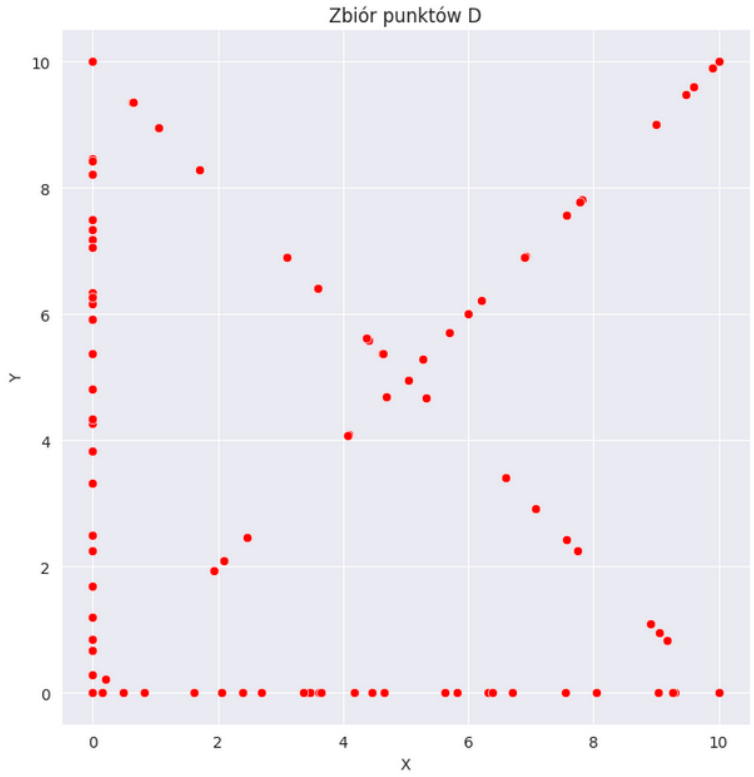
Rysunek 1



Rysunek 2



Rysunek 3



Rysunek 4

4.2 Modyfikowanie funkcji generującej zbiory

Zgodnie z treścią zadania funkcje generujące punkty na płaszczyźnie zostały zmodyfikowane aby umożliwić następujące zmiany w parametrach:

a) liczba punktów, przedziały dla współrzędnych,

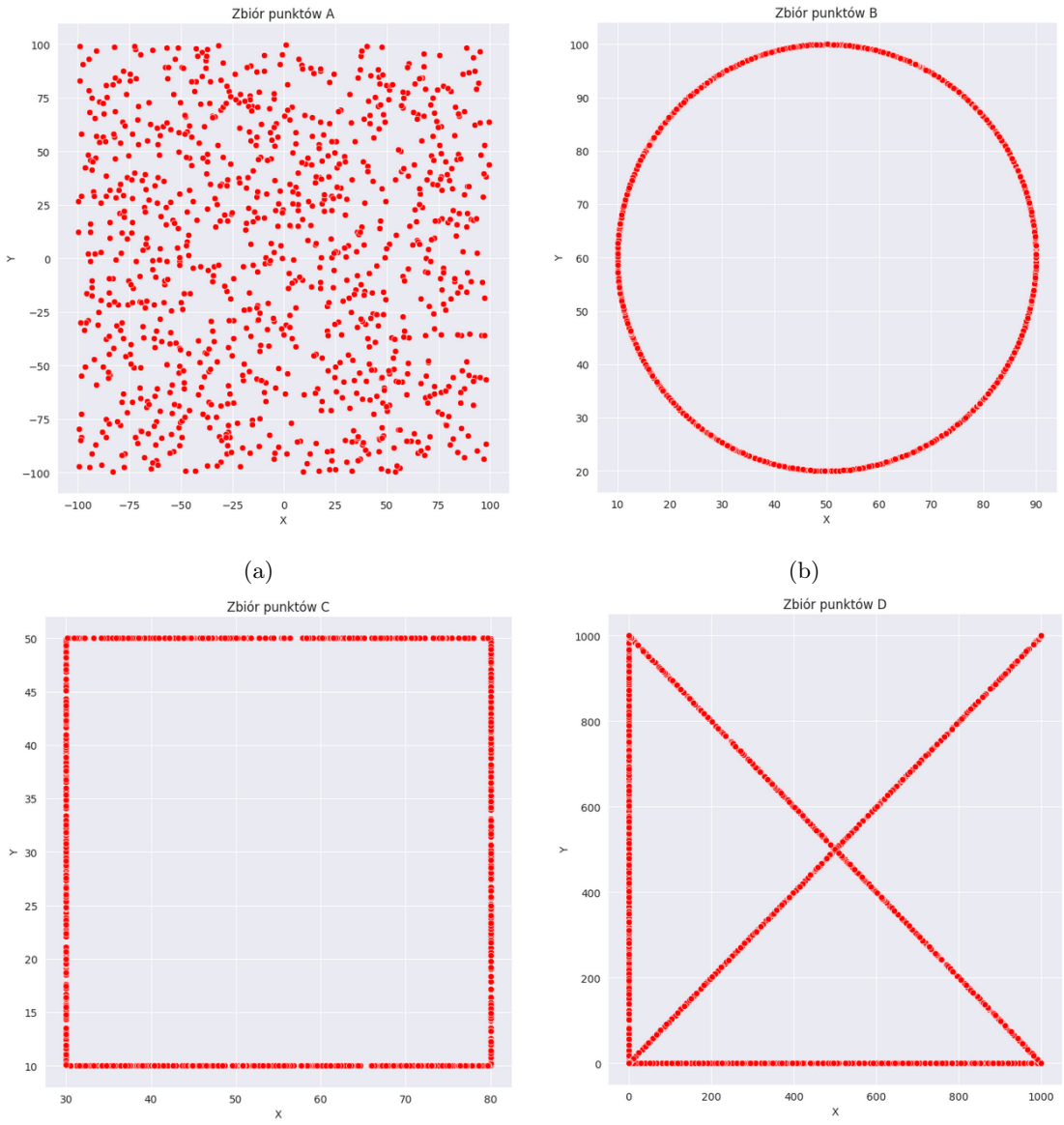
b) liczba punktów, środek i promień okręgu,

c) liczba punktów, wierzchołki prostokąta,

d) wierzchołki kwadratu, liczba punktów na osiach, liczba punktów na przekątnych.

4.3 Wizualizacja zbiorów dla nowych parametrów

Wizualizacja ma na celu potwierdzenie iż punkty są generowane w odpowiednich przedziałach, dlatego liczba punktów które zostały wygenerowane jest znacznie większa od tych podanych w zad1.



(a) (b)

(c) (d)

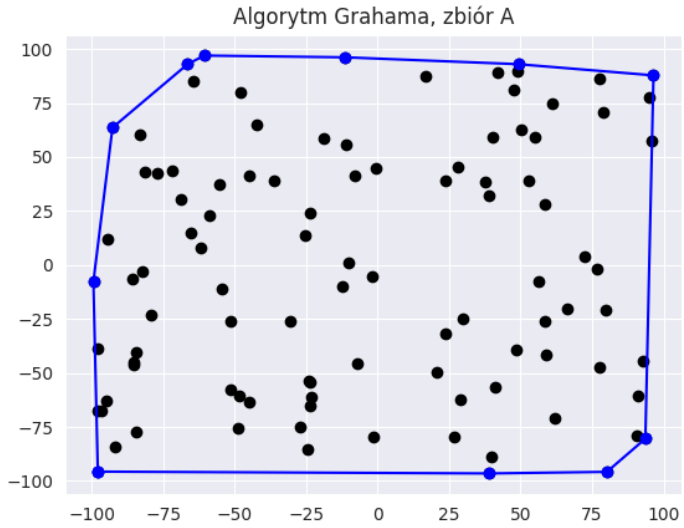
Rysunek 5

5 Wizualizacja algorytmów

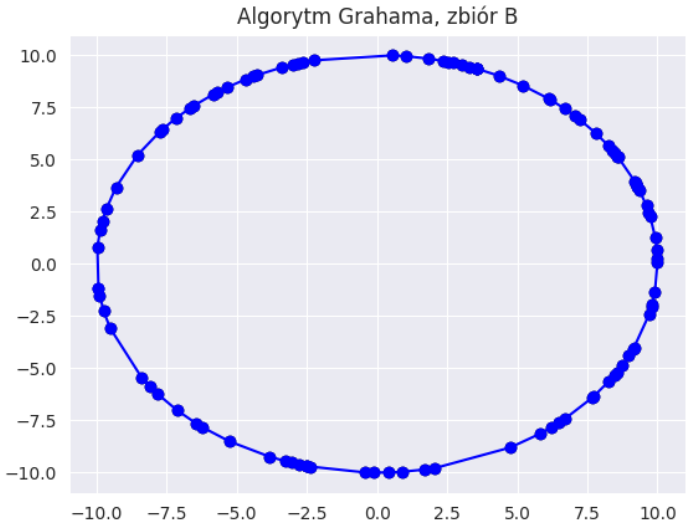
Algorytmy zostały uruchomione na tej samej jednostce obliczeniowej oraz na tym samym środowisku. W funkcji *orient* podczas obliczania wyznacznika macierzy 2x2 za epsilon została przyjęta wartość 10^{-12}.

5.1 Algorytm Grahama

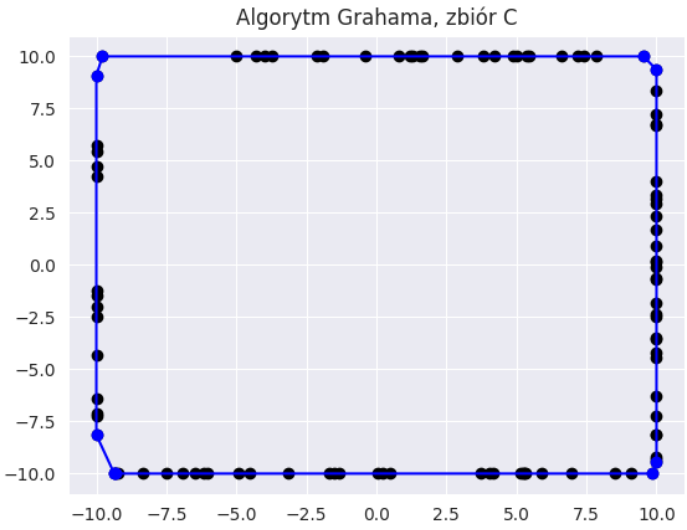
Wizualizacja otoczki dla poszczególnych zbiorów punktów z *zad*1.



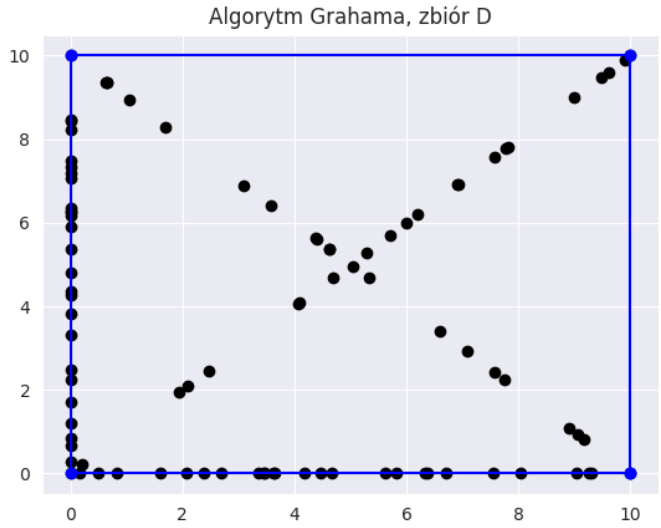
Rysunek 6



Rysunek 7



Rysunek 8

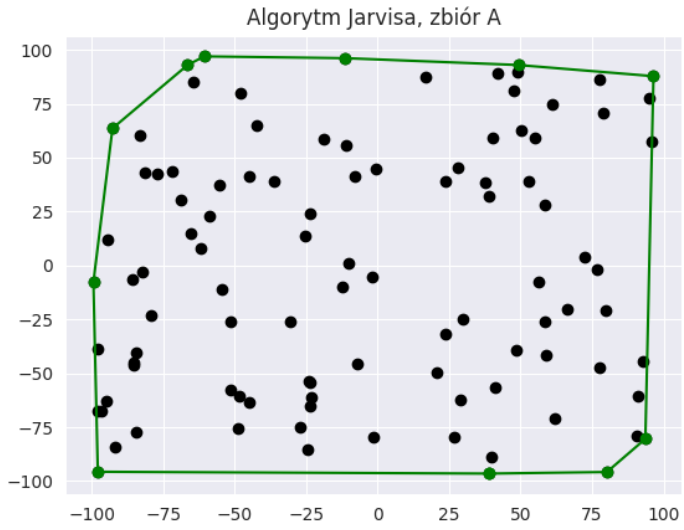


Rysunek 9

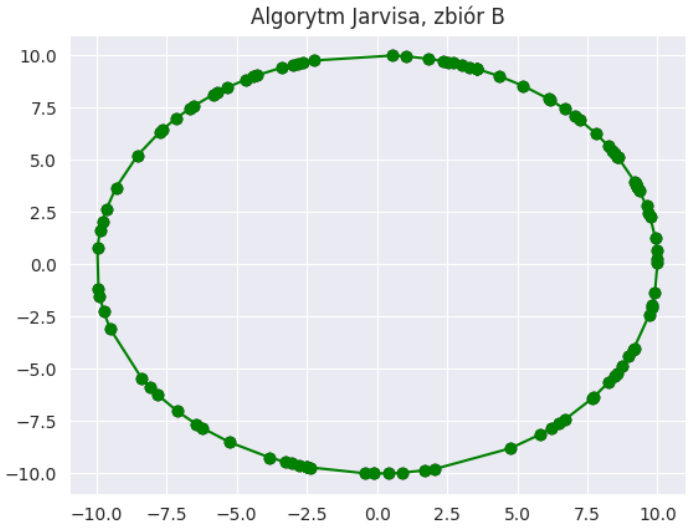
W zborze D algorytm Grahama radzi sobie bardzo dobrze, ponieważ pomimo współliniowości niektórych punktów, jedyne punkty które należą do otoczki znajdują się na krawędziach kwadratu.

5.2 Algorytm Jarvisa

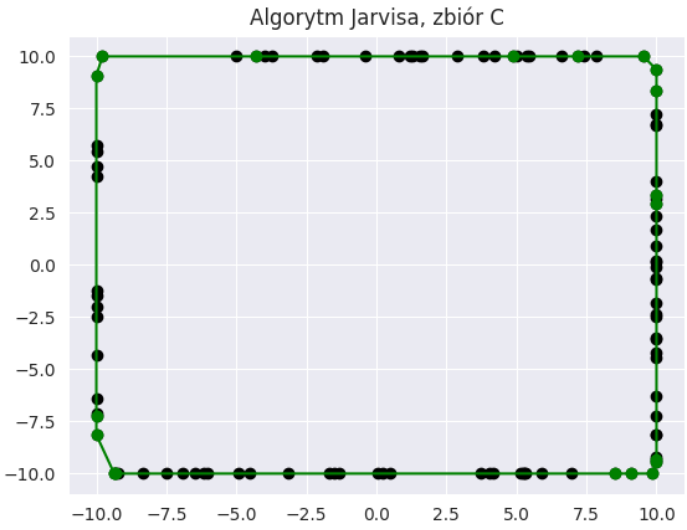
Wizualizacja otoczki dla poszczególnych zbiorów punktów z *zad*1.



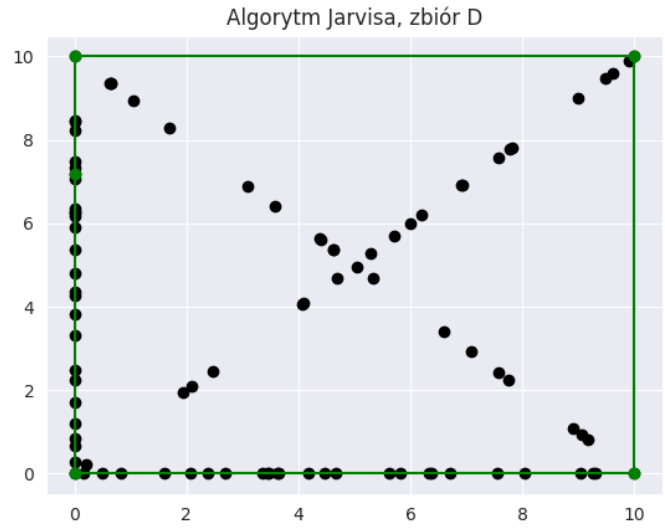
Rysunek 10



Rysunek 11



Rysunek 12



Rysunek 13

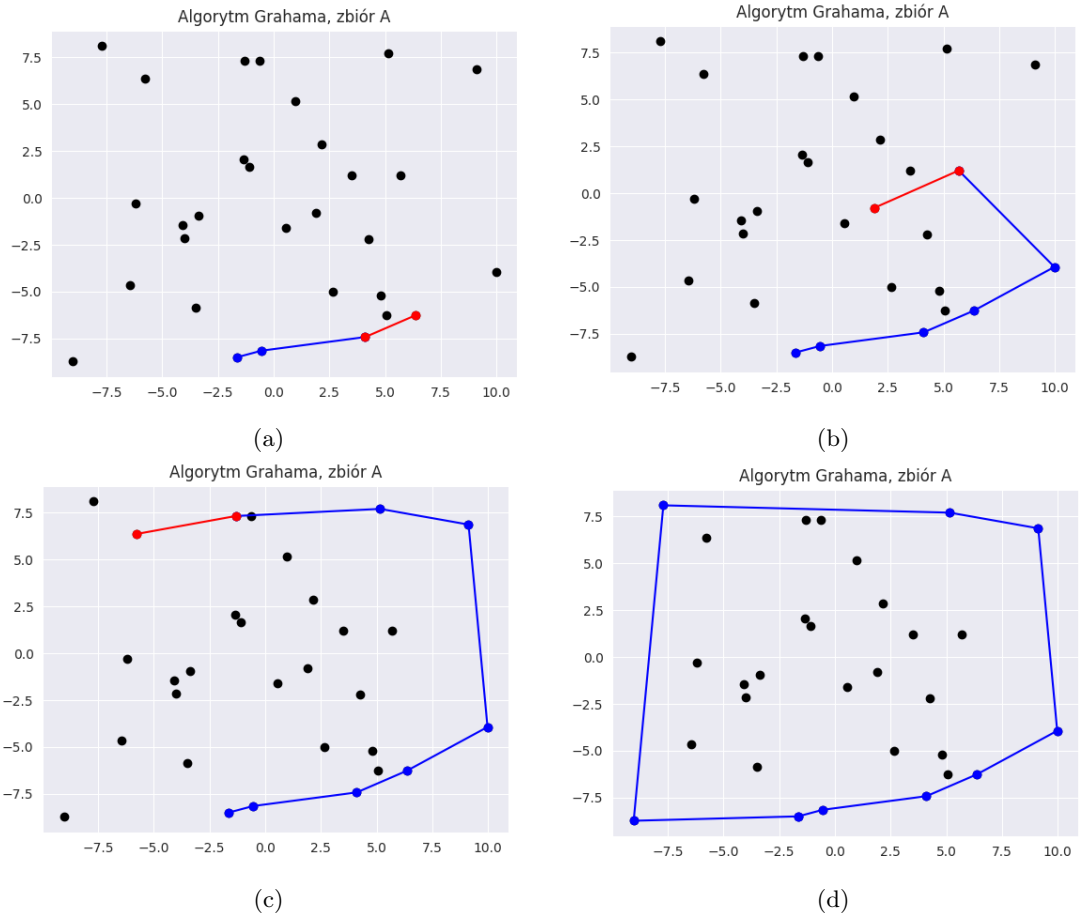
Niestety w zbiorze testowym D algorytm Jarvisa sprawdził się znacznie gorzej, z uwagi iż przydzielił do otoczki punkty współliniowe.

5.3 Wizualizacja graficzna działania algorytmów

Zgodnie z treścią zadania zaimplementowana została również wizualizacja działania poszczególnych algorytmów dla wszystkich zbiorów. Funkcja posiada argument: *show_steps* domyślnie ustawiony na wartość fałszywą, lecz po zmienieniu pozwala zwizualizować powstawanie otoczki wypukłej. Z uwagi na dość obszerną ilość obrazków ilustrujących kolejne kroki, zbiory z podpunktu pierwszego zostały utworzone ponownie lecz ze zmniejszoną ilość generowanych punktów. Zostały do tego

użyte funkcje napisane w *zad*2.
To właśnie na nich została zostało przedstawione działanie algorytmów.

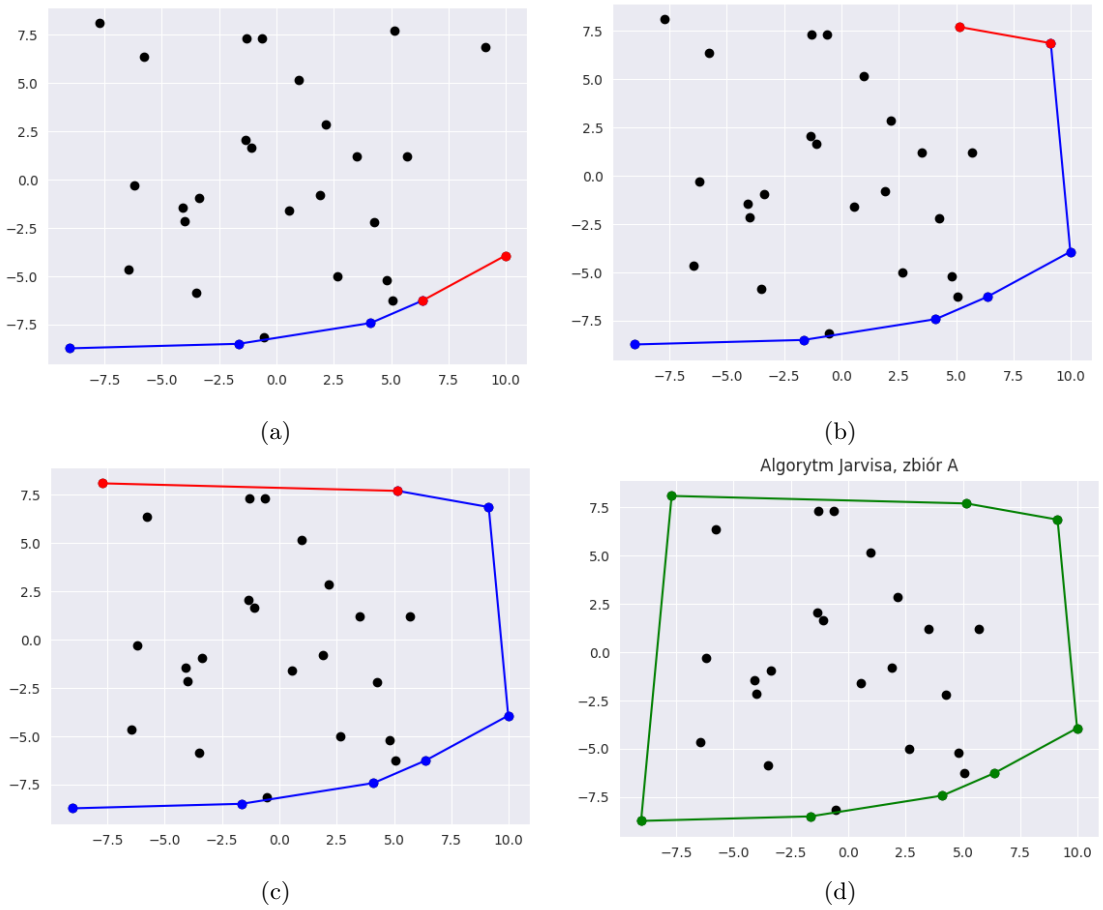
Przykładowa wizualizacja algorytmu Grahama dla pomniejszonego zbioru A:



(a) (b) (c) (d)

Rysunek 14

Przykładowa wizualizacja algorytmu Jarvisa dla pomniejszonego zbioru A:



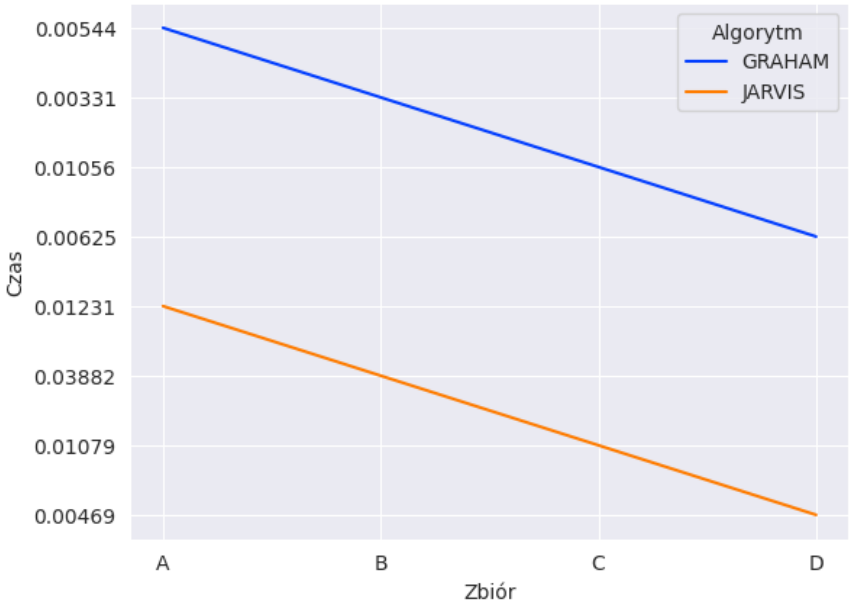
(a) (b)

(c) (d)

Rysunek 15

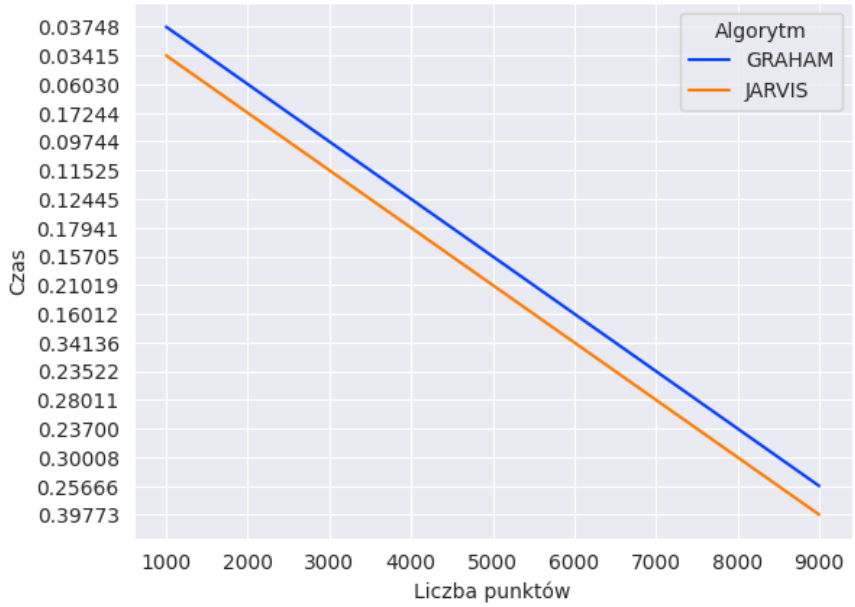
5.4 Porównanie czasów

Dla zbioru punktów z zadania pierwszego:



Rysunek 16

Czas w zależności od ilości punktów na zbiorze A:



Rysunek 17

6 Wnioski

Ćwiczenie miało na celu porównanie wyników algorytmów, a także czasów w których dane algorytmy działają dla różnych zbiorów punktów.
Podczas testowania programów zauważyłem, że czasami algorytm Grahama w zbiorze A niepoprawnie przydziela jeden punkt za dużo do otoczki przez co wielokąt który tworzą wyróżnione punkty nie jest wypukły. Sytuacja ta przydarzyła się jeden raz, lecz ciężko jest mi stwierdzić czy nie jest to błąd w implementacji algorytmu.
Problem ten nie miał miejsca w algorytmie Jarvisa, gdzie punkty wyznaczone do otoczki zawsze tworzyły wielokąt wypukły.

Dla zbioru B obydwa algorytmy zachowały się podobnie przyporządkowując do otoczki wszystkie punkty znajdujące się na okręgu. Wynik ten był do przewidzenia z uwagi iż rozmiar okręgu jest zbyt mały aby punkty mogły zostać przyporządkowane za współliniowe, zaś w innym przypadku pominięcie jakiegoś punktu sprawiłoby, iż nie należałby on do wnętrza otoczki.

Zaproponowane zbiory punktów miały na celu przetestowanie czy algorytmy poprawnie pomijają punkty współliniowe przyporządkowując do otoczki tylko minimalną ilość punktów potrzebnych do zawarcia całych figur. Tutaj na znaczną przewagę wychodzi algorytm owijania prezentu który dla zbioru D w którym punkty są generowane również na samych wierzchołkach kwadratu przyporządkowywał tylko niezbędne punkty.
Niestety sytuacja była inna w dla algorytmu Grahama w którego wyniku dla zbioru D znajdowało się więcej niż 4 punkty.

W czasie działania algorytmów na znaczną przewagę wyszedł algorytm Jarvisa.