

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 przedstawienie graficzne działania algorytmów: Grahama 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  $p_0, p_1, p_2, \dots, p_m$ .
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( $p_i$ ),  $i \leftarrow i+1$ 
    else
        pop(s)
```

Do wyznaczenia kąta kąta pomiędzy punktami została wykorzystana funkcja *orient*:

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

### 3.1.2 Złożoność obliczeniowa

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

## 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  $\neq i$  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
         $i \leftarrow k$ 
until  $i = i_0$ 
```

### 3.2.2 Złożoność obliczeniowa

Złożoność jest rzędu  $O(n^2)$ .

---

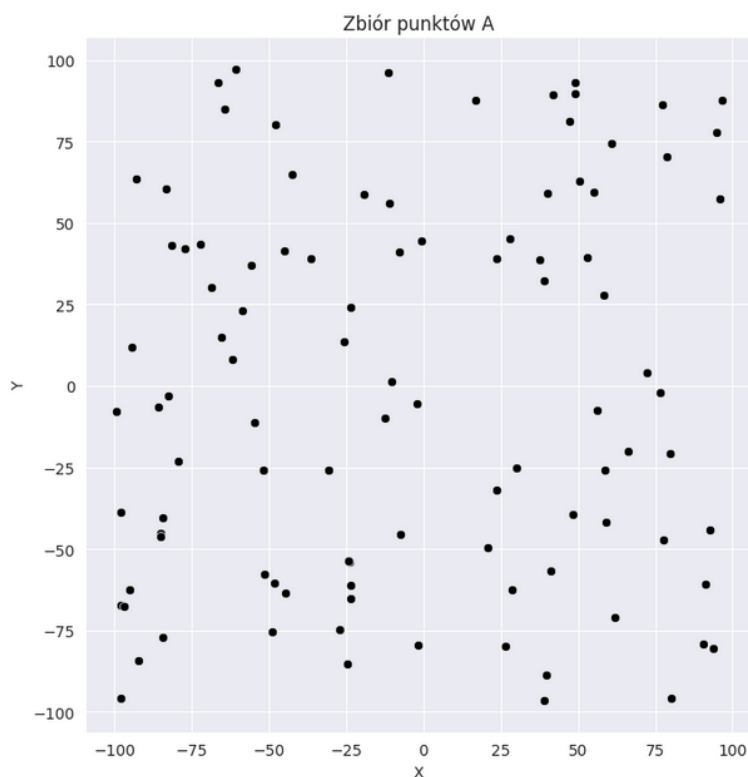
## 4 Generacja punktów na płaszczyźnie

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

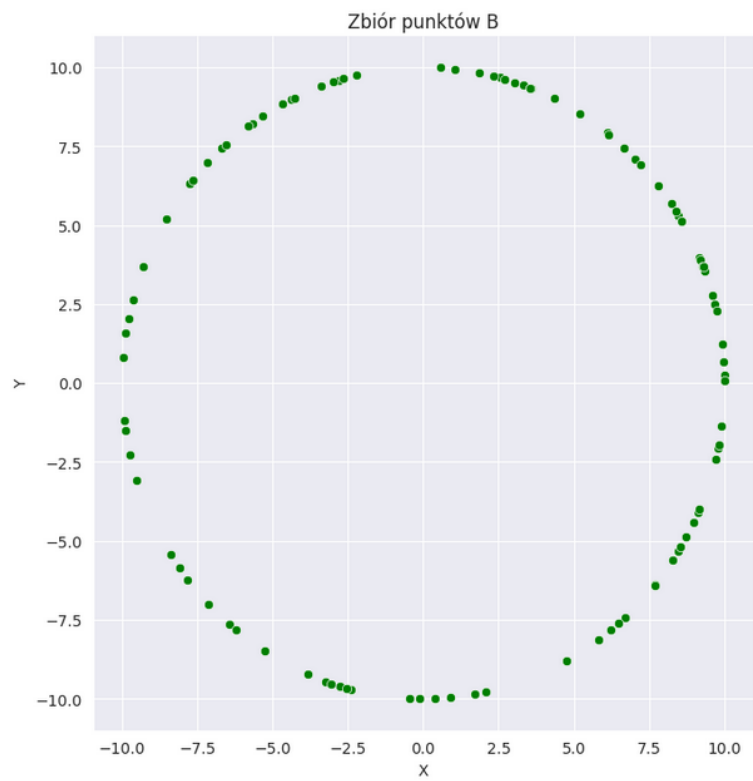
- a) zbiór zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału  $[-100, 100]$ ,
- b) zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku  $(0,0)$  i promieniu  $R=10$ ,
- c) zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach  $(-10, 10)$ ,  $(-10,-10)$ ,  $(10,-10)$ ,  $(10,10)$ ,
- 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.

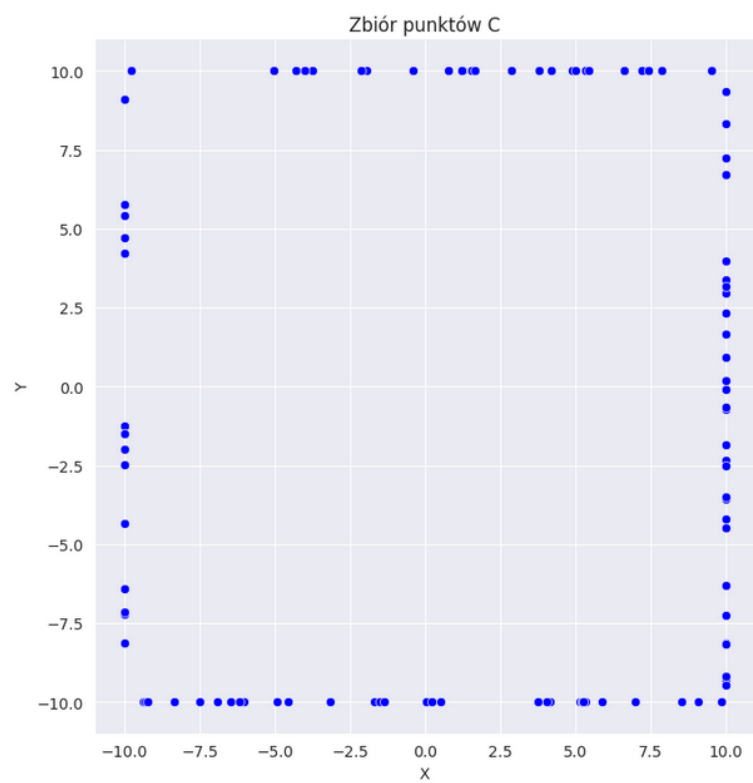
### 4.1 Wizualizacja graficzna zbiorów:



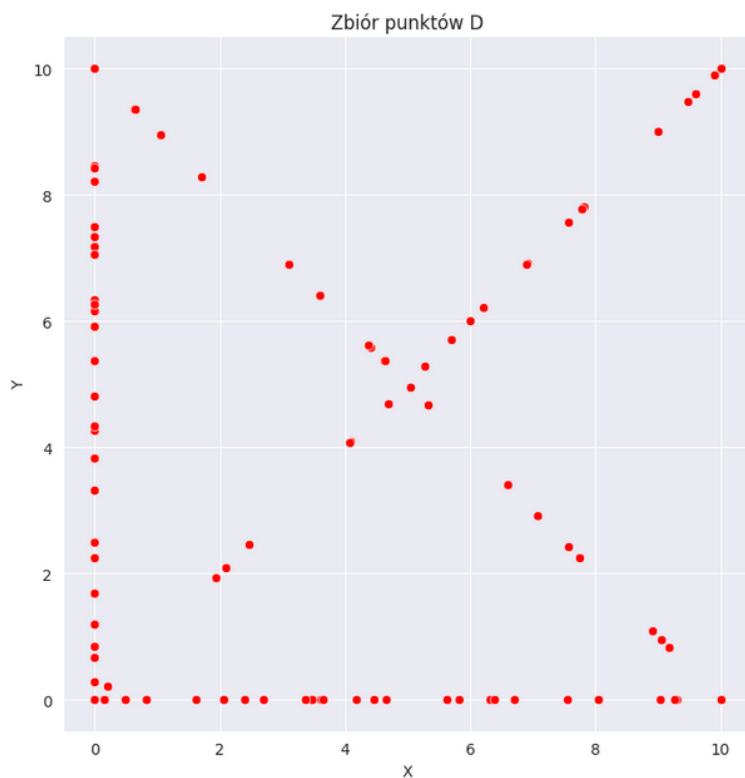
Rysunek 1



Rysunek 2



Rysunek 3



Rysunek 4

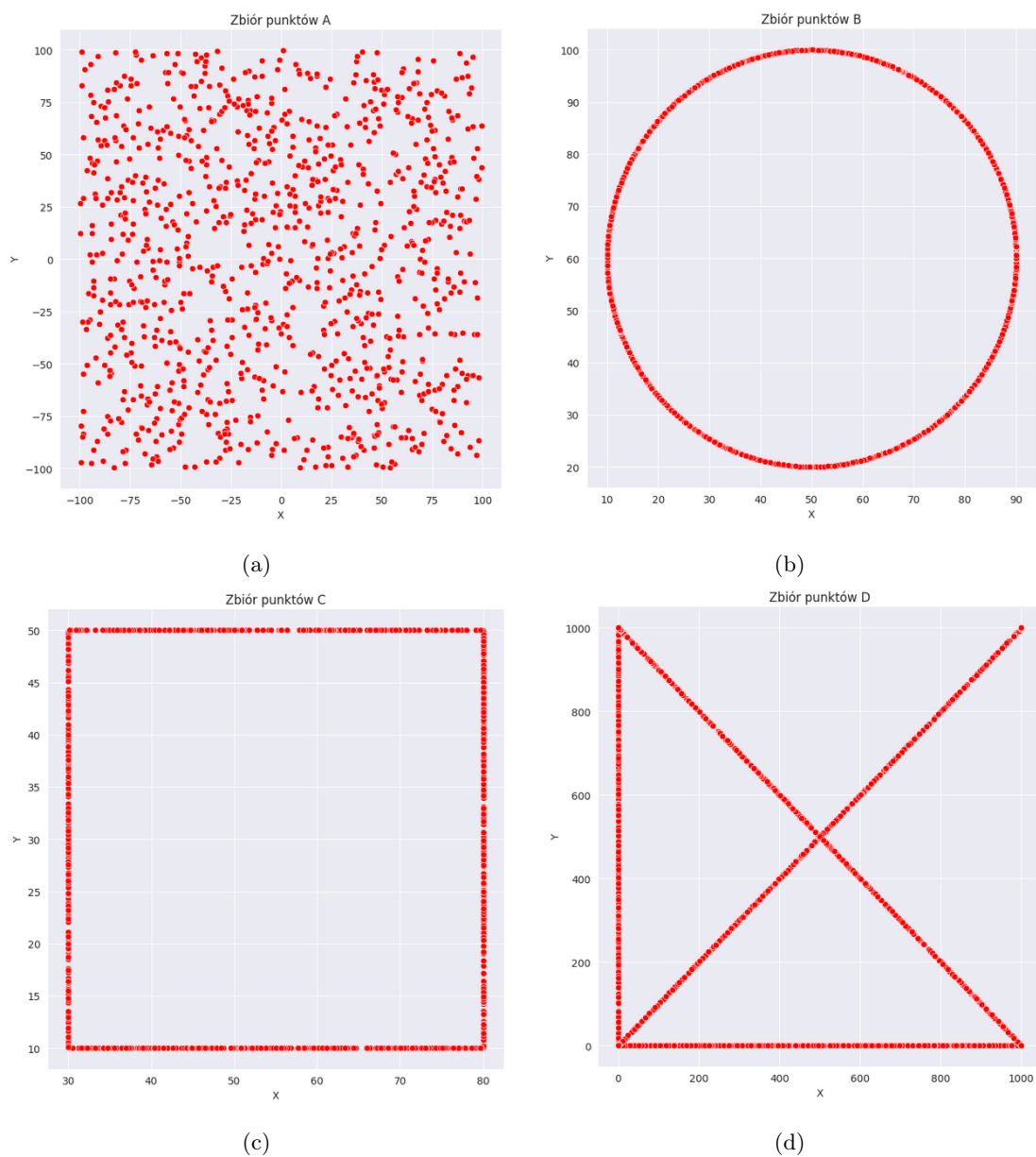
## 4.2 Modyfikowanie funkcji generującej zbiory

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.



Rysunek 5

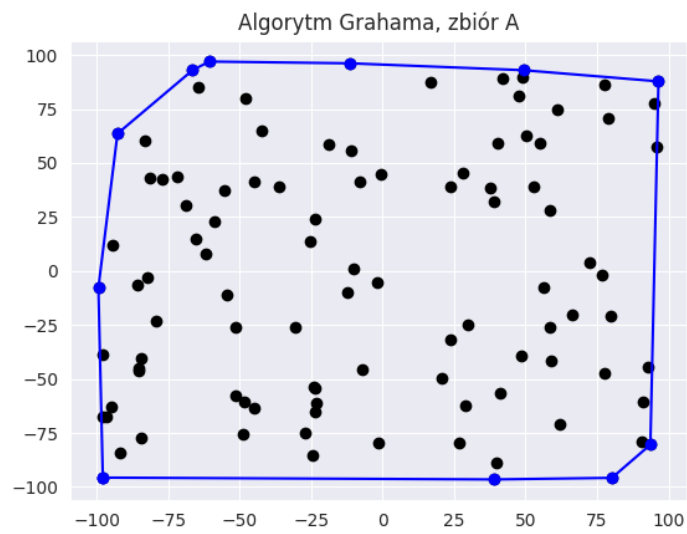
## 5 Wizualizacja algorytmów

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  $10^{-12}$ .

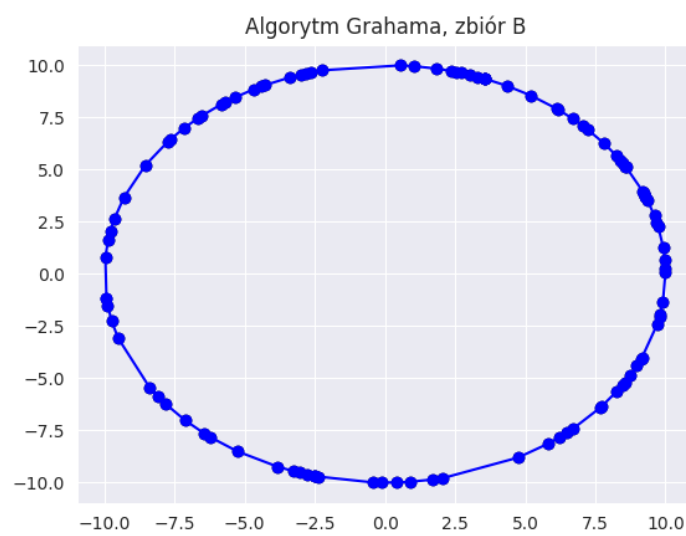
---

## 5.1 Algorytm Grahama

Wizualizacja otoczki dla poszczególnych zbiorów punktów z *zad1*.

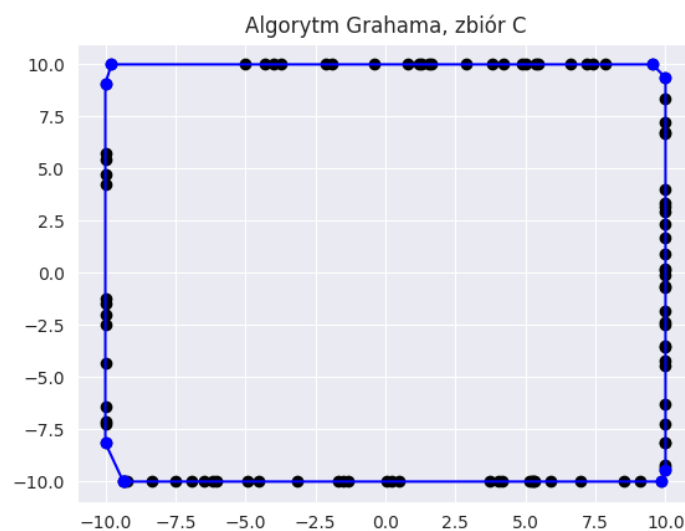


Rysunek 6

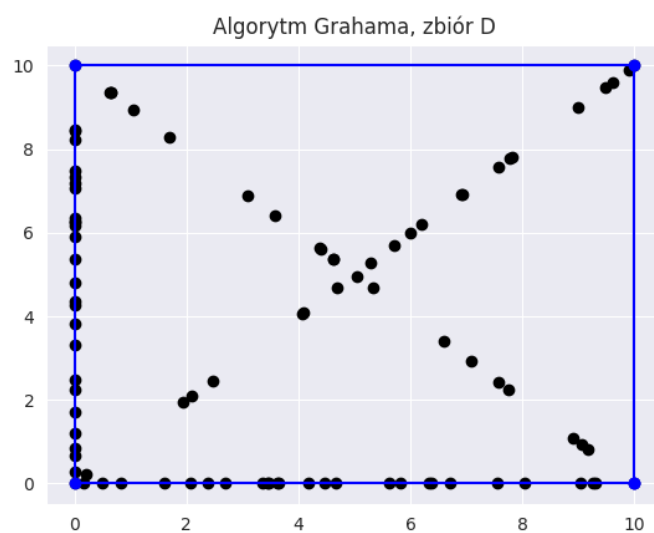


Rysunek 7





Rysunek 8

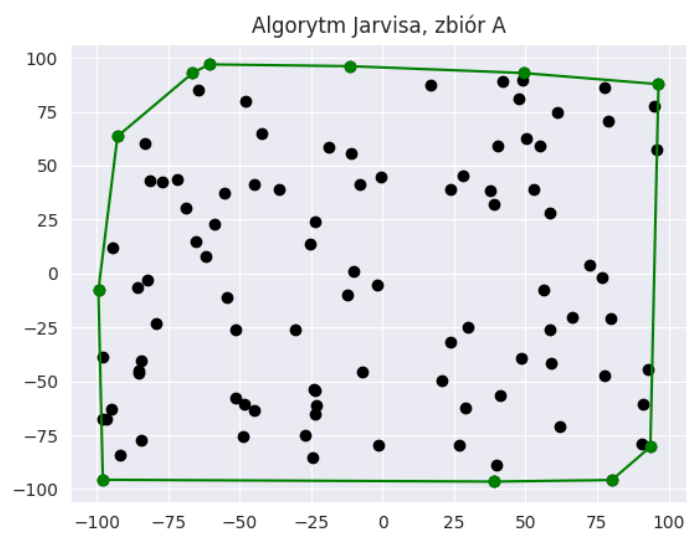


Rysunek 9

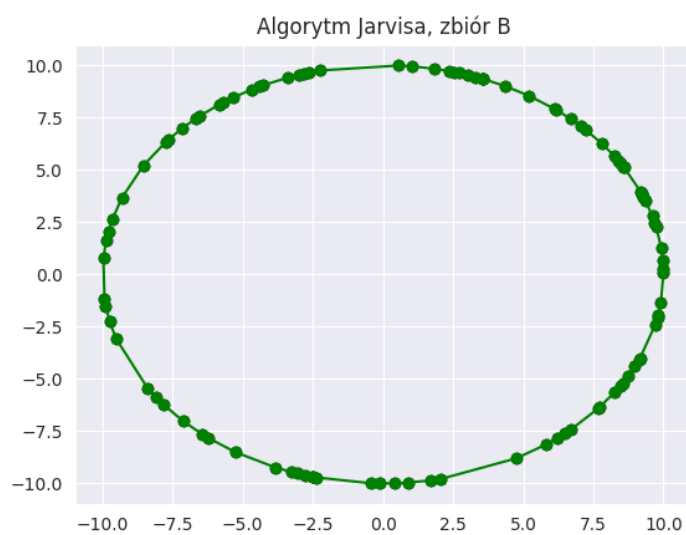
W zbiorze D algorytm Grahama radzi sobie bardzo dobrze, ponieważ pomimo współliniowości niektórych punktów, jedyne punkty które należą do otoczki znajdują się na krawędziach kwadratu.

## 5.2 Algorytm Jarvisa

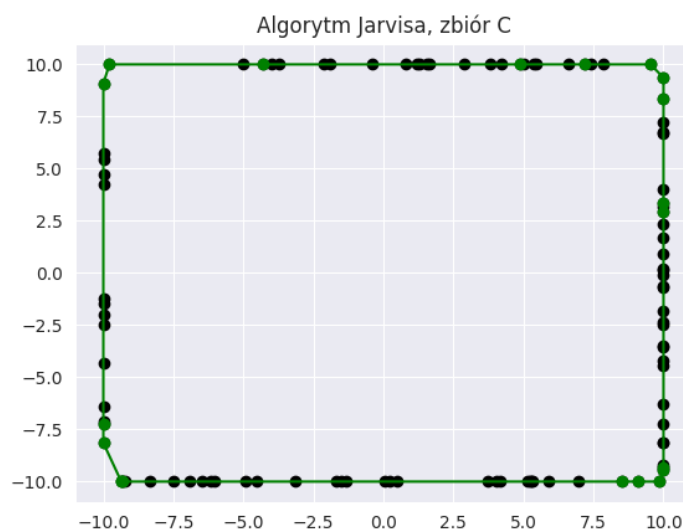
Wizualizacja otoczki dla poszczególnych zbiorów punktów z *zad1*.



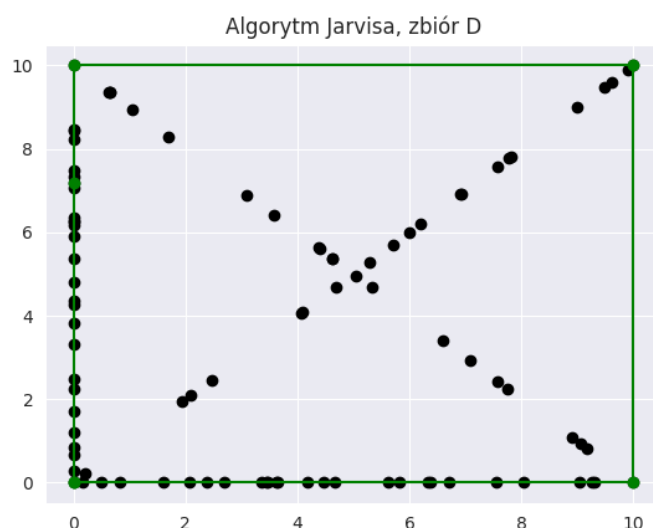
Rysunek 10



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.

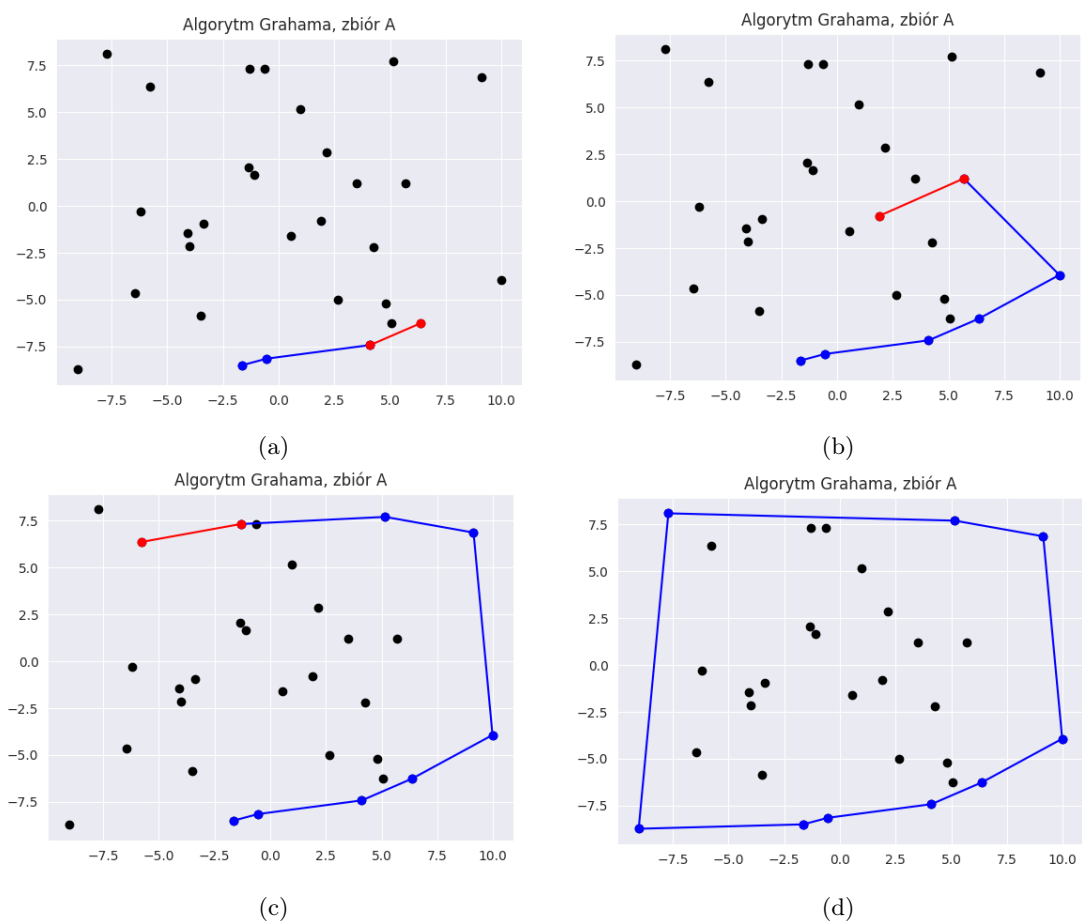
### 5.3 Wizualizacja graficzna działania algorytmów

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 zmianieniu 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 *zad2*.

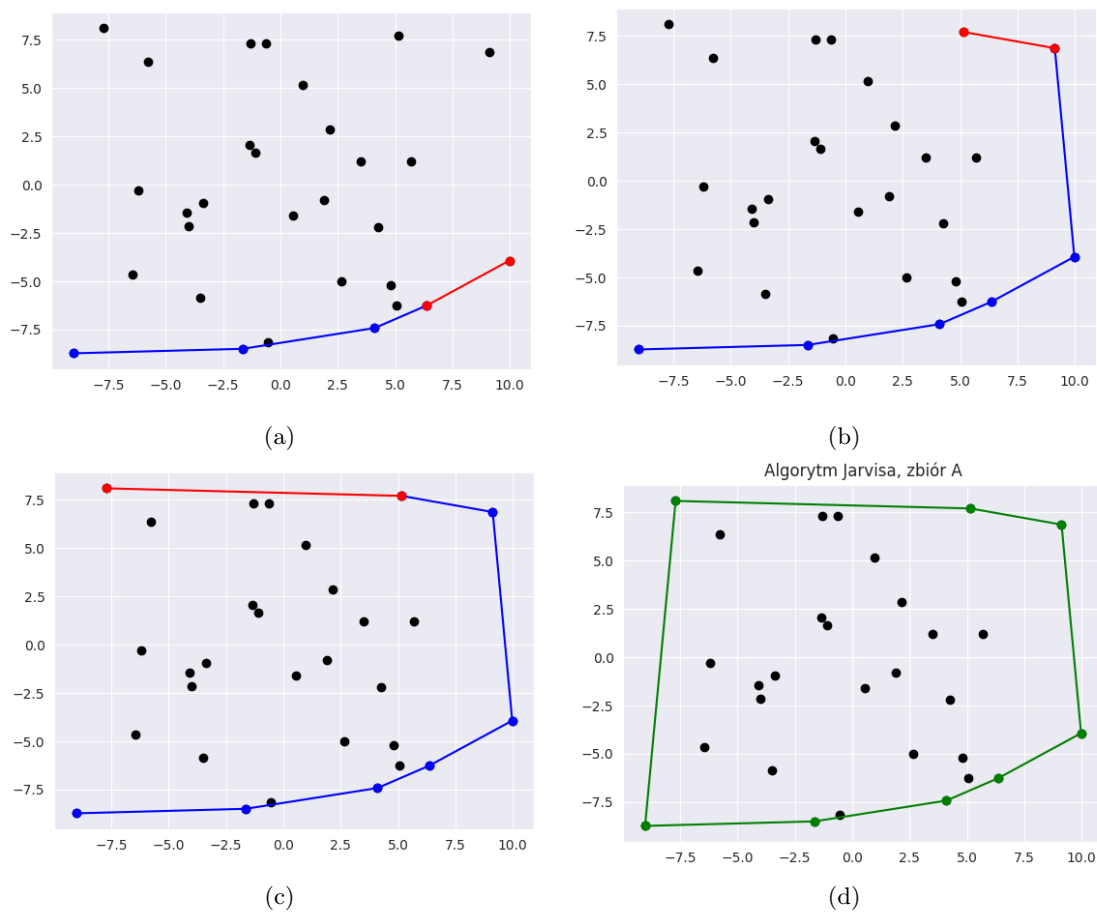
To właśnie na nich została została przedstawione działanie algorytmów.

Przykładowa wizualizacja algorytmu Grahama dla pomniejszonego zbioru A:



Rysunek 14

Przykładowa wizualizacja algorytmu Jarvisa dla pomniejszonego zbioru A:



Rysunek 15

---

## 6 Porównanie czasów działania algorytmów

### 6.1 Zbiory A, B, C, D z zadania pierwszego

	Zbiór	Graham [s]	Jarvis [s]	Szybszy
0	A	0.00347	0.00203	Algorytm Jarvisa
1	B	0.00151	0.02566	Algorytm Grahama
2	C	0.00513	0.00180	Algorytm Jarvisa
3	D	0.00470	0.00064	Algorytm Jarvisa

Rysunek 16

### 6.2 Zbiór A

	Liczba punktów w zbiorze A	Graham [s]	Jarvis [s]	Szybszy
0	10000	0.34796	0.51415	Algorytm Grahama
1	9000	0.31830	0.27136	Algorytm Jarvisa
2	8000	0.23176	0.24755	Algorytm Grahama
3	7000	0.27312	0.19590	Algorytm Jarvisa
4	6000	0.21216	0.13445	Algorytm Jarvisa
5	5000	0.17044	0.13674	Algorytm Jarvisa
6	4000	0.13978	0.12025	Algorytm Jarvisa
7	3000	0.10113	0.06256	Algorytm Jarvisa
8	2000	0.11382	0.11395	Algorytm Grahama
9	1000	0.04096	0.04638	Algorytm Grahama

Rysunek 17

---

### 6.3 Zbiór B

	Liczba punktów w zbiorze B	Graham [s]	Jarvis [s]	Szybszy
0	1000	0.02557	2.12235	Algorytm Grahama
1	900	0.02441	1.26373	Algorytm Grahama
2	800	0.02011	1.22310	Algorytm Grahama
3	700	0.01322	0.57494	Algorytm Grahama
4	600	0.04132	0.59836	Algorytm Grahama
5	500	0.01128	0.30549	Algorytm Grahama
6	400	0.00848	0.20287	Algorytm Grahama
7	300	0.00542	0.12332	Algorytm Grahama
8	200	0.00262	0.04339	Algorytm Grahama
9	100	0.00132	0.01101	Algorytm Grahama

Rysunek 18

### 6.4 Zbiór C

	Liczba punktów w zbiorze C	Graham [s]	Jarvis [s]	Szybszy
0	10000	0.59308	0.65180	Algorytm Grahama
1	9000	0.39762	0.58737	Algorytm Grahama
2	8000	0.32026	0.45763	Algorytm Grahama
3	7000	0.34388	0.36426	Algorytm Grahama
4	6000	0.23636	0.31133	Algorytm Grahama
5	5000	0.21006	0.17593	Algorytm Jarvisa
6	4000	0.14626	0.11227	Algorytm Jarvisa
7	3000	0.09589	0.14953	Algorytm Grahama
8	2000	0.06283	0.10785	Algorytm Grahama
9	1000	0.02603	0.02713	Algorytm Grahama

Rysunek 19

---

## 6.5 Zbiór D

	Liczba punktów w zbiorze D	Graham [s]	Jarvis [s]	Szybszy
0	30000	1.71844	0.27551	Algorytm Jarvisa
1	27000	1.48620	0.42640	Algorytm Jarvisa
2	24000	1.28890	0.30494	Algorytm Jarvisa
3	21000	1.12111	0.21906	Algorytm Jarvisa
4	18000	1.37529	0.17209	Algorytm Jarvisa
5	15000	0.60100	0.16952	Algorytm Jarvisa
6	12000	0.56981	0.15569	Algorytm Jarvisa
7	9000	0.54830	0.09585	Algorytm Jarvisa
8	6000	0.39592	0.19677	Algorytm Jarvisa
9	3000	0.26626	0.10406	Algorytm Jarvisa

Rysunek 20

## 7 Wnioski

Ćwiczenie miało na celu porównanie wyników algorytmów, a także czasów w których dane algorytmy działają dla różnych zbiorów punktów.

W zbiorze punktów A obydwa algorytmy poprawnie przyporządkowały dane punkty do otoczki wypukłej, zaś pod względem czasowym wypadły bardzo podobnie.

Dla zbioru B algorytmy zachowały się identycznie 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. Czasowo algorytm Grahama sprawdził się tutaj znacznie lepiej.

Dla zbioru C lepiej sprawdził się algorytm Grahama.

Odwrotna sytuacja nastąpiła w zbiorze D w którym algorytm Jarvisa okazał się znacząco szybszy. 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 Grahama 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 Jarvisa w którego wyniku dla zbioru D znajdowało się 5 punktów.