Algorytmy geometryczne
Labolatorium nr 2

Patryk Klatka
Grupa nr 4

10 listopada 2022

1 Opis ćwiczenia

Ćwiczenie polegało na zaimplementowaniu dwóch podstawowych algorytmów do wyznaczania otoczki wypukłej, a następnie na przetestowaniu ich na różnych zestawach punktów. Otoczka wypukła, to najmniejszy możliwy zbiór wypukły, który będzie zawierał dany podzbiór. W praktyce wyznaczane są punkty (w naszym przypadku, punkty w $\mathbb{R}^2$), które po połączeniu będą tworzyły figurę wypukłą, która będzie zawierać wszystkie punkty ze zbioru. Algorytmy, które zostały wybrane do implementacji, to algorytm Grahama, działający w złożoności obliczeniowej $O(n \log n)$, oraz algorytm Jarvisa, działający w złożoności obliczeniowej $O(nk)$, gdzie k, to liczba punktów otoczki (zatem złożoność może się zwiększyć do $O(n^2)$). Na potrzeby ćwiczenia, algorytmy zostały zmodyfikowane tak, aby można było rozważać zbiory z punktami współliniowymi (więcej niż 3 punkty współliniowe). Dodatkowo, aby uniknąć problemów z precyzją obliczeniową, zamiast obliczania kątów, jakie tworzą wektory z osią OX jest sprawdzane położenie danego punktu względem wektora. Do badania położenia punktu, wykorzystywany jest do tego wyznacznik 3x3 własnej implementacji, zaimplementowany wykorzystując metodę Sarrusa z tolerancją dla zera `1e-10`.

1.1 Algorytm Grahama

Pseudokod:

1. Znajdź punkt p_0 o najmniejszej współrzędnej y, a jeżeli istnieje kilka takich punktów, to dodatkowo o najmniejszej współrzędnej x.
2. Usuń p_0 z listy z punktami *dataset*.
3. Posortuj *dataset* względem najmniejszego punktu w porządek leksykograficzny: kąt jaki tworzy wektor $\vec{p_0p}$ z osią OX zaczynającą się w p_0, a jeżeli jest kilka punktów, które mają ten sam kąt, to po odległościach rosnąco (możemy skorzystać z metryki Euklidesowej, Czebyszewa lub taksówkowej, ponieważ badamy punkty na tej samej prostej).
4. Do posortowanego *dataset* dodajemy na początek punkt p_0.
5. Do stosu s dodajemy trzy pierwsze punkty z *dataset*.
   5.1. Jeżeli punkty są współliniowe, to usuwamy środkowy punkt z s.
6. Dopóki nie przejrzymy wszystkich punktów w *dataset*:
   6.1. Jeżeli badany punkt s_0 leży na lewo od wektora $\vec{s_2s_1}$, gdzie s_1 oraz s_2 to pierwszy i drugi element stosu, to dodajemy go do s.
   6.2. Jeżeli badany punkt s_0 jest współliniowy z wektorem $\vec{s_2s_1}$, to zamieniamy pierwszy punkt stosu na punkt, który leży dalej od punktu s_2.
   6.3. W przeciwnym przypadku usuwamy punkt s_1 ze stosu.

Złożoność: $O(n \log n)$.
Modyfikacje podstawowego algorytmu w punktach: 5.1, 6.2.

1.2 Algorytm Jarvisa

Pseudokod:

1. Znajdź punkt p_{min} o najmniejszej współrzędnej y, a jeżeli istnieje kilka takich punktów, to dodatkowo o najmniejszej współrzędnej x.
2. Niech p_1 to dowolny punkt z listy z punktami *dataset* różny od p_{min} oraz $p_0 = p_{min}$.
3. Wykonuj:
   3.1. Dla każdego punktu $p_2 \neq p_0$:
      3.1.1. Jeżeli p_2 leży na prawo od wektora $p_0\vec{p}_1$ to $p_1 = p_2$ lub jeżeli p_2 jest współliniowy z wektorem $p_0\vec{p}_1$ oraz `distance`$(p_0p_1) <$ `distance`(p_0p_2), to $p_1 = p_2$
   3.2. Znaleziona krawędź (p_0, p_1) to część otoczki wypukłej.
   3.3. Niech $p_0 = p_1$.
   3.4. Jeżeli p_0 jest równe p_{min}, to wyjdź z pętli.

Złożoność: $O(nk)$, gdzie k, to liczba punktów otoczki wypukłej.
Modyfikacje podstawowego algorytmu w punktach: 3.1.1.

2 Użyte narzędzia i środowisko pracy

Cały program został napisany w języku Python za pomocą popularnej, interaktywnej platformy Jupyter Notebook. W narzędziu graficznym zostały wykorzystywane biblioteki takie jak `matplotlib` (do graficznego przedstawiania danych) oraz `numpy` (do generowania zestawów danych). W otrzymanym narzędziu graficznym został zmieniony backend biblioteki matplotlib (`%matplotlib widget`) poprzez doinstalowanie biblioteki `ipympl`, aby narzędzie to mogło obsługiwać nowsze edytory, m.in. `VSCode`. Dodatkowo, dodana została możliwość tworzenia tytułów wykresów/podwykresów.

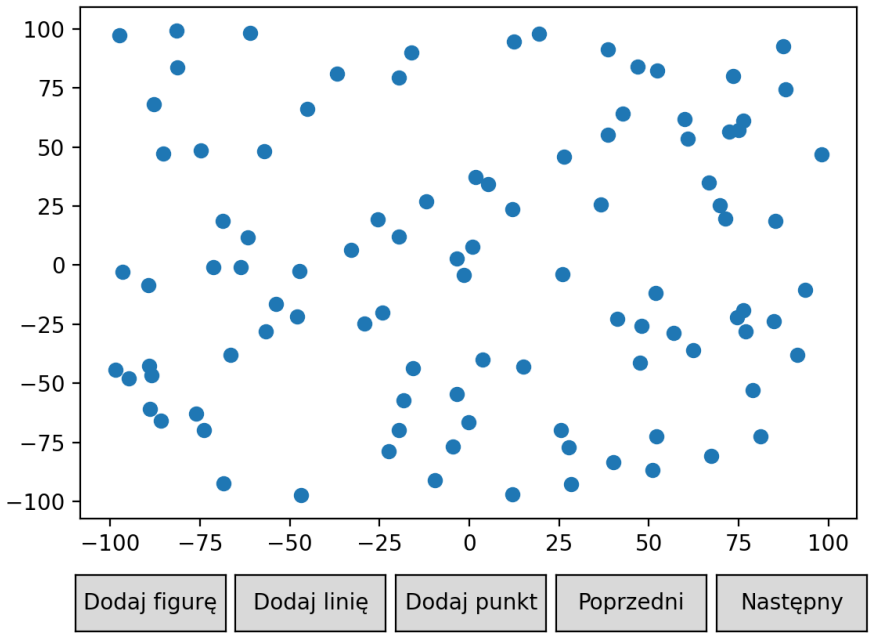
Dane sprzętowe:

- System Windows 10 x64
- Procesor Intel Core i5-8250U
- RAM 8GB
- Wersja Python: 3.10

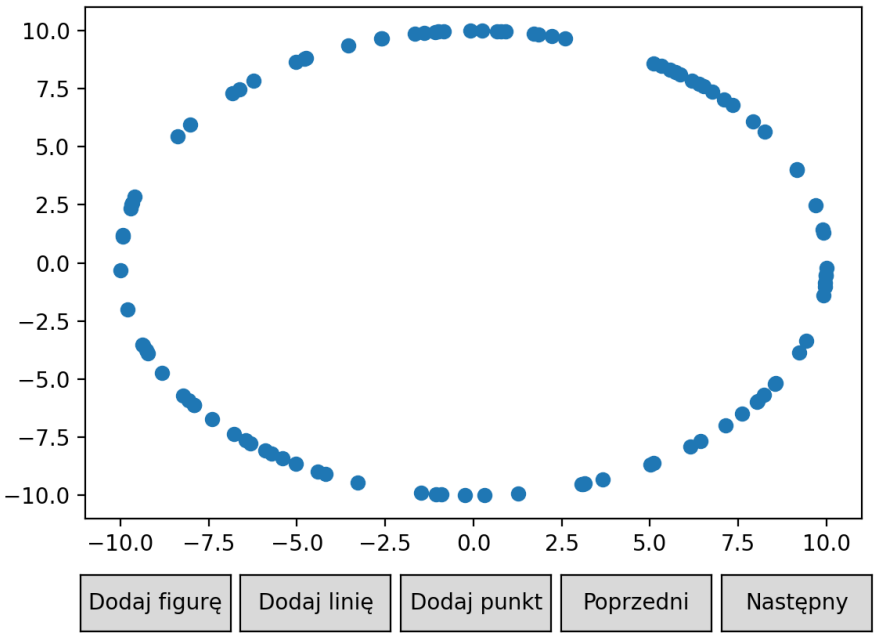
3 Generacja zestawów punktów

W ramach ćwiczenia należało wygenerować cztery zbiory punktów:

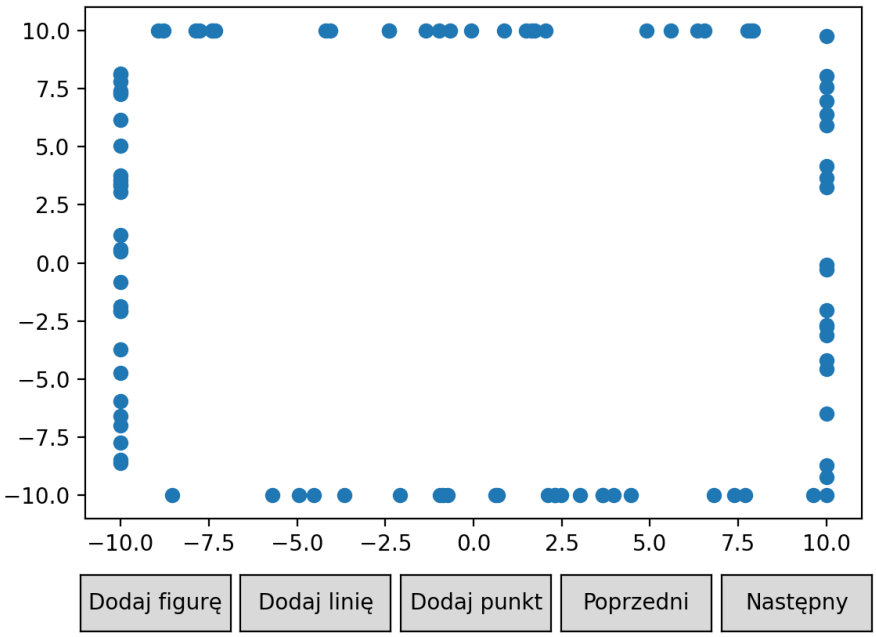
- Zestaw danych A: zbiór zawierający 100 losowo wygenerowanych punktów o współrzędnych z przedziału $[-100, 100]$
- Zestaw danych B: zbiór zawierający 100 losowo wygenerowanych punktów leżących na okręgu o środku $(0, 0)$ i promieniu $R = 10$
- Zestaw danych C: zbiór zawierający 100 losowo wygenerowanych punktów leżących na bokach prostokąta o wierzchołkach $(-10, 10)$, $(-10, -10)$, $(10, -10)$, $(10, 10)$
- Zestaw danych D: zbiór 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.



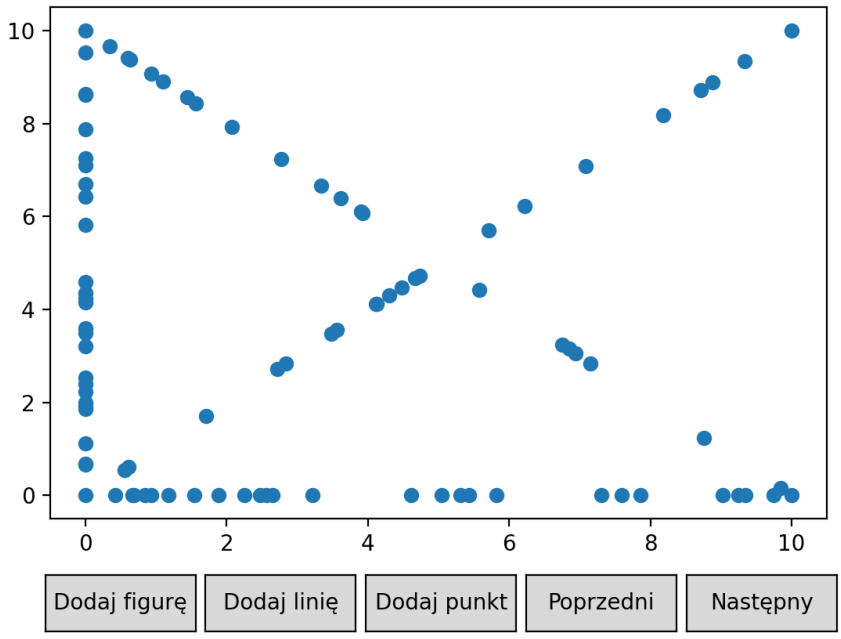
Rysunek 1: Zestaw danych A



Rysunek 2: Zestaw danych B



Rysunek 3: Zestaw danych C



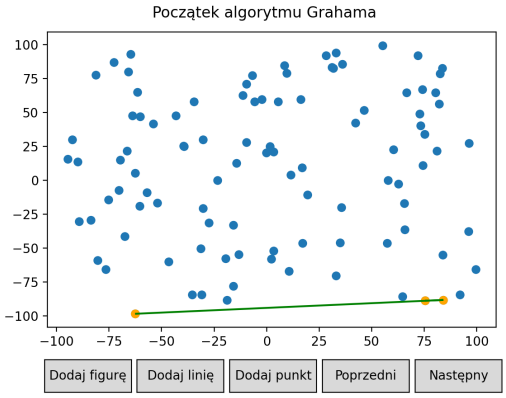
Rysunek 4: Zestaw danych D

4 Wizualizacje działania algorytmów

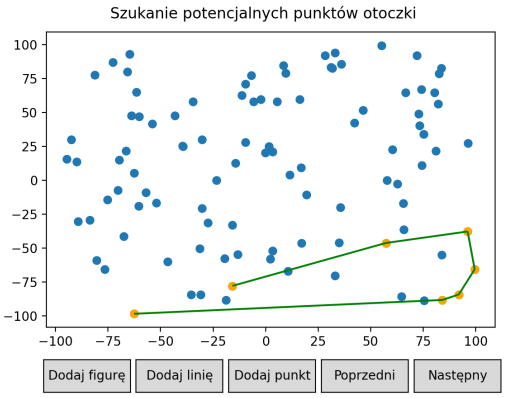
W ramach ćwiczenia, zaimplementowana została wizualizacja poszczególnych kroków algorytmów znajdujących otoczkę wypukłą.

4.1 Algorytm Grahama

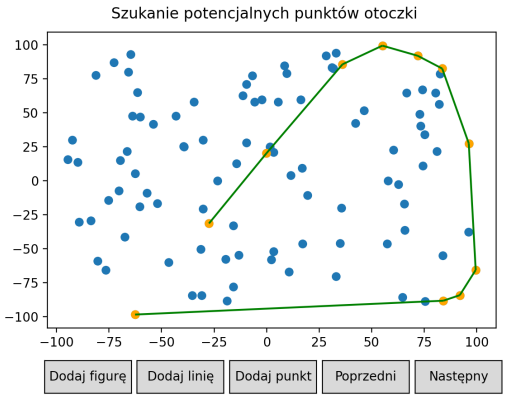
4.1.1 Zestaw danych A



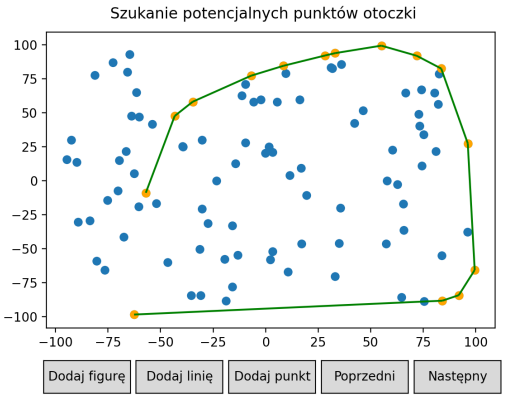
(a) Krok 1



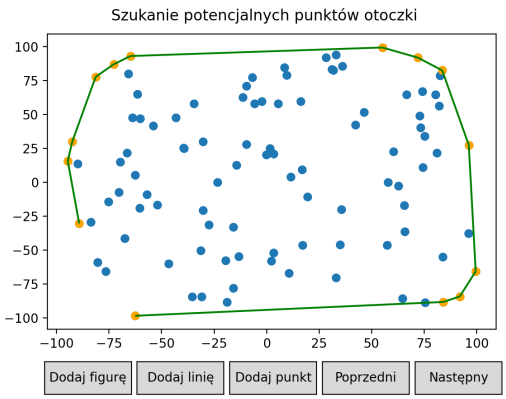
(b) Krok 2



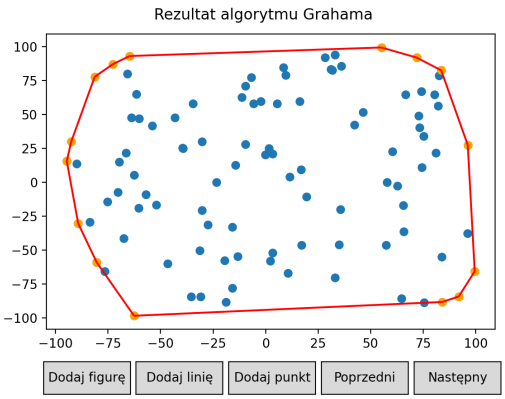
(c) Krok 3



(d) Krok 4



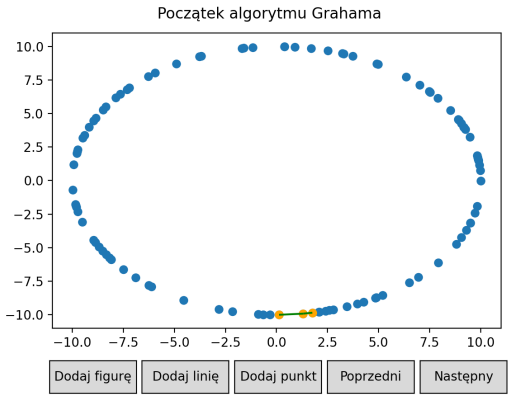
(e) Krok 5



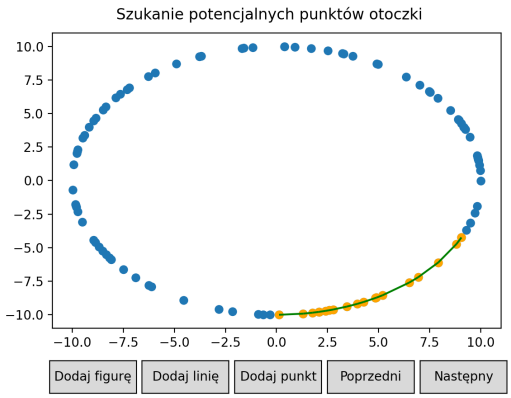
(f) Krok 6

Rysunek 5: Wizualizacja algorytmu Grahama dla zestawu danych A

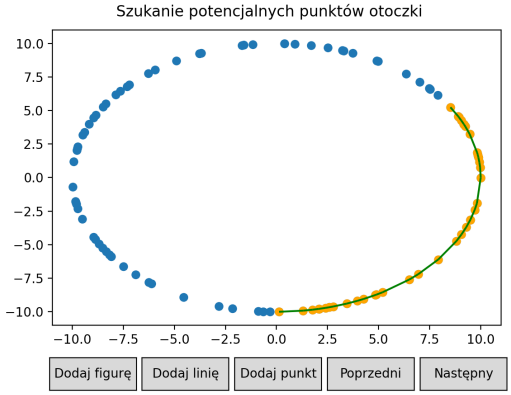
4.1.2 Zestaw danych B



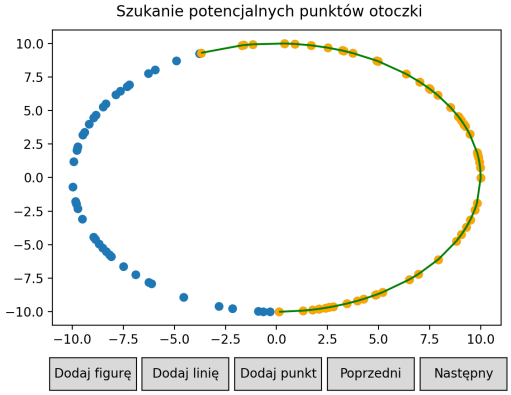
(a) Krok 1



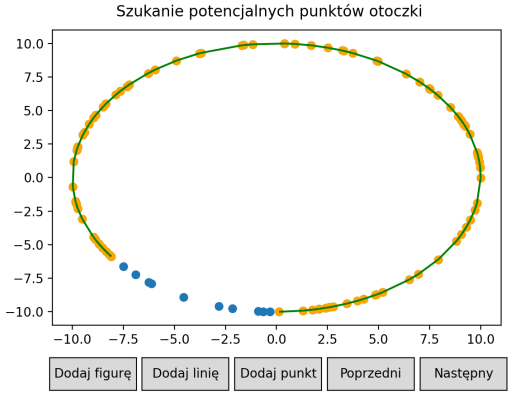
(b) Krok 2



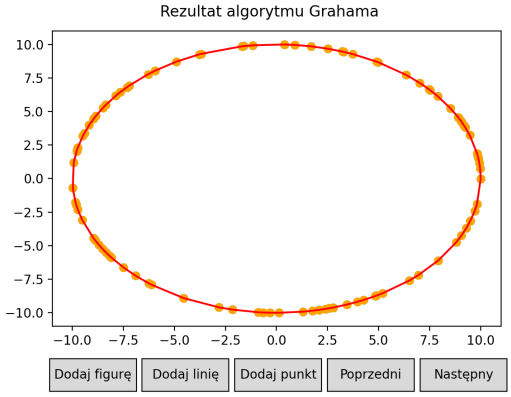
(c) Krok 3



(d) Krok 4



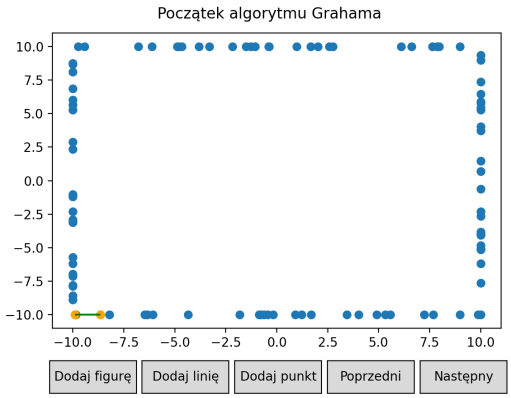
(e) Krok 5



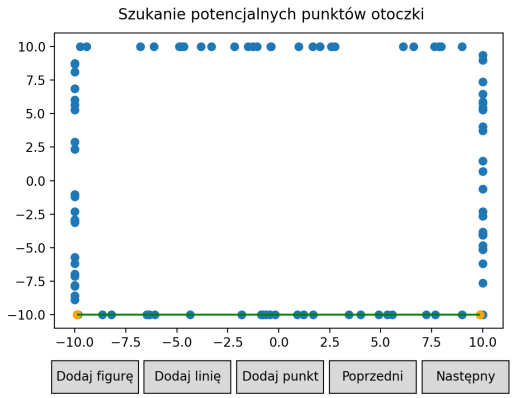
(f) Krok 6

Rysunek 6: Wizualizacja algorytmu Grahama dla zestawu danych B

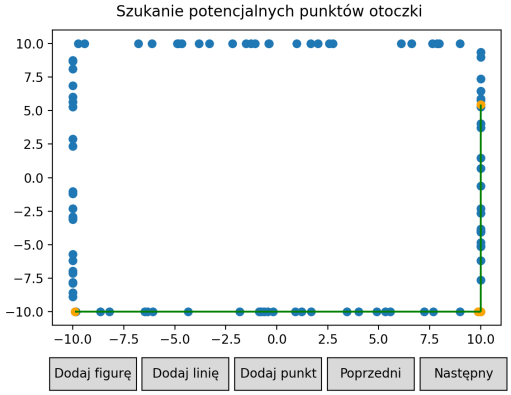
4.1.3 Zestaw danych C



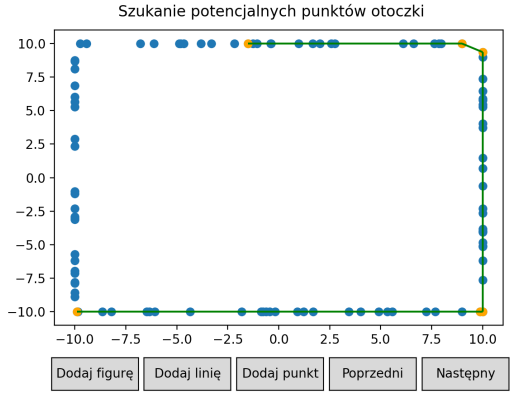
(a) Krok 1



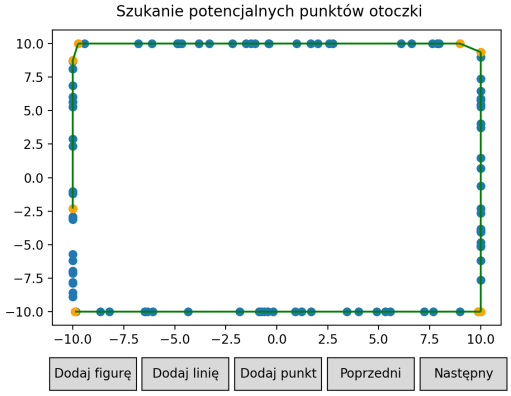
(b) Krok 2



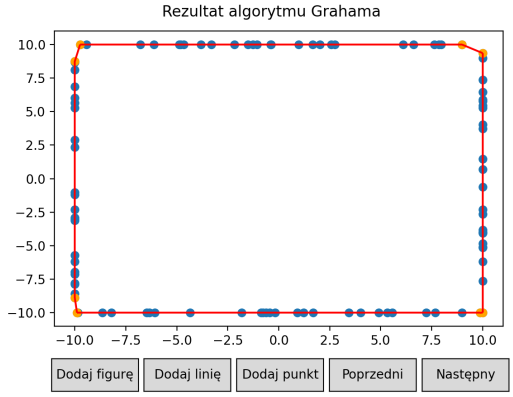
(c) Krok 3



(d) Krok 4



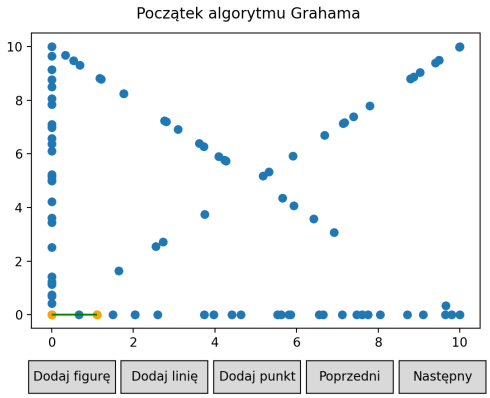
(e) Krok 5



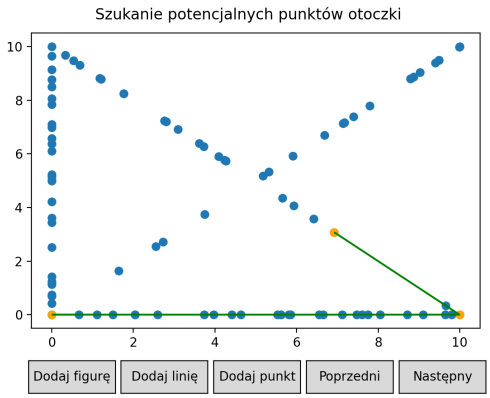
(f) Krok 6

Rysunek 7: Wizualizacja algorytmu Grahama dla zestawu danych C

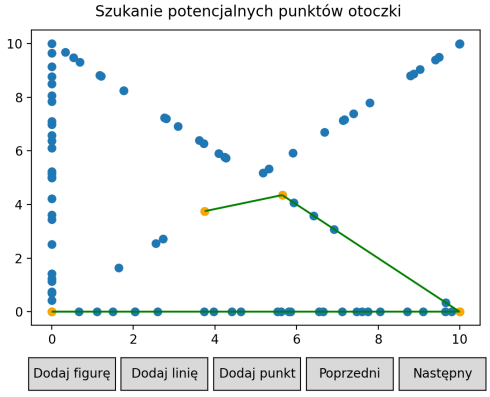
4.1.4 Zestaw danych D



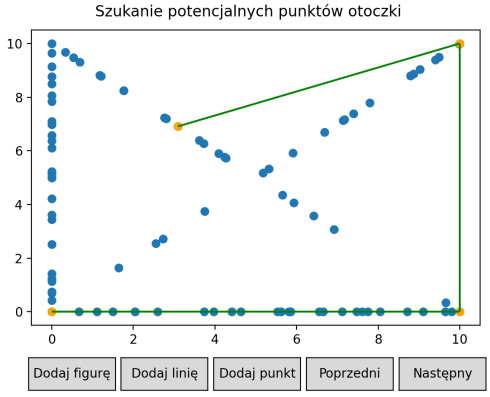
(a) Krok 1



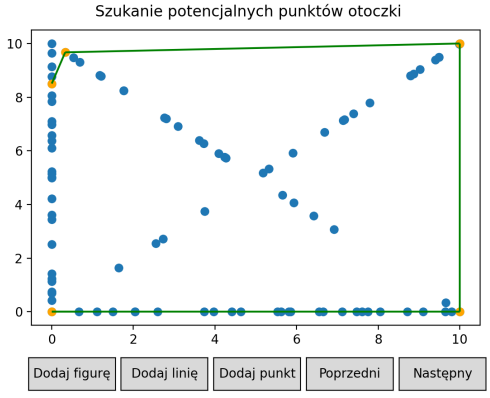
(b) Krok 2



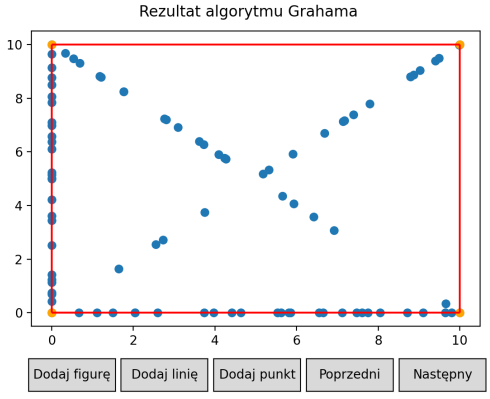
(c) Krok 3



(d) Krok 4



(e) Krok 5

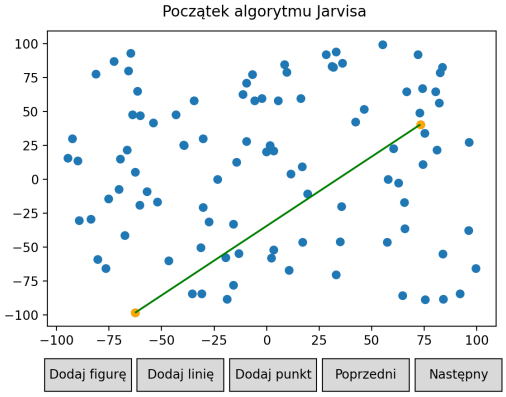


(f) Krok 6

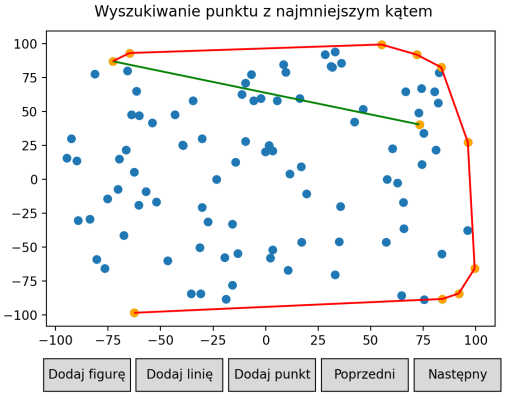
Rysunek 8: Wizualizacja algorytmu Grahama dla zestawu danych D

4.2 Algorytm Jarvisa

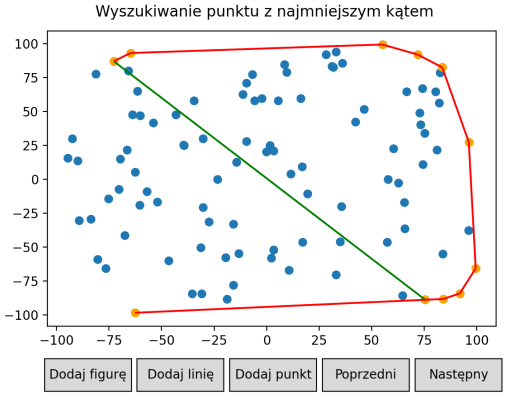
4.2.1 Zestaw danych A



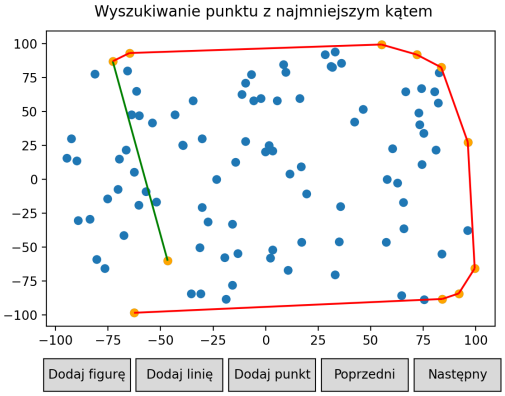
(a) Krok 1



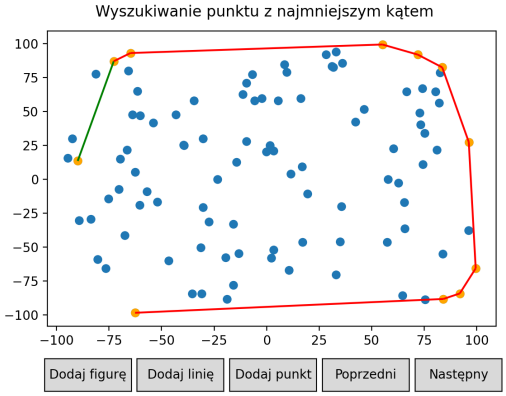
(b) Krok 2



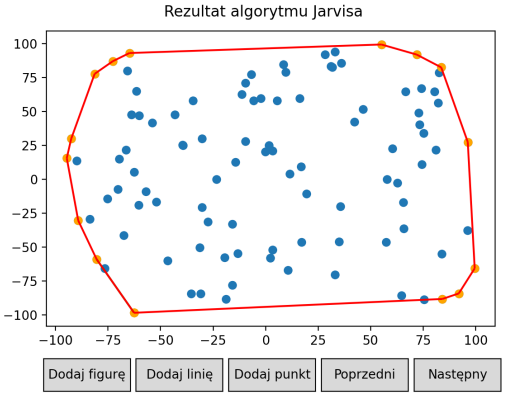
(c) Krok 3



(d) Krok 4



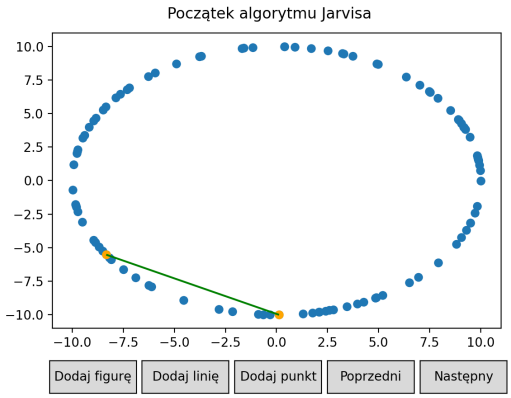
(e) Krok 5



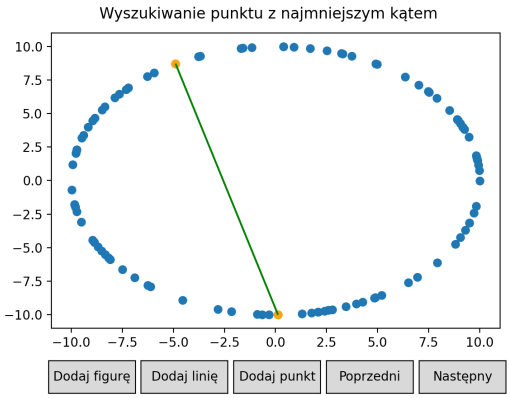
(f) Krok 6

Rysunek 9: Wizualizacja algorytmu Jarvisa dla zestawu danych A

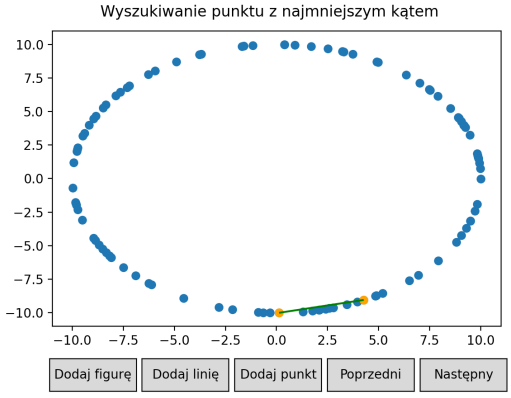
4.2.2 Zestaw danych B



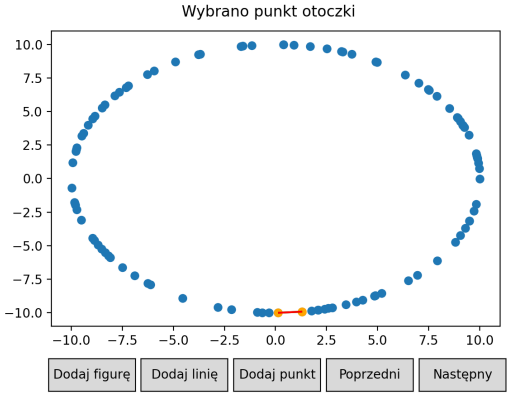
(a) Krok 1



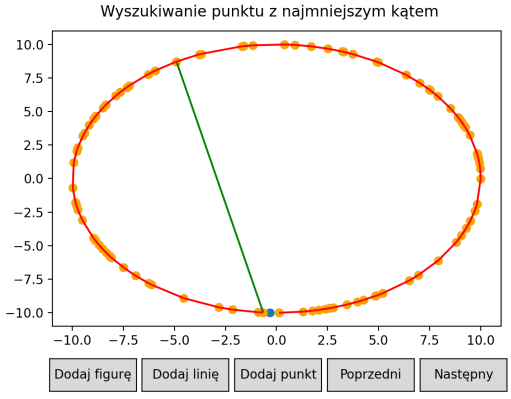
(b) Krok 2



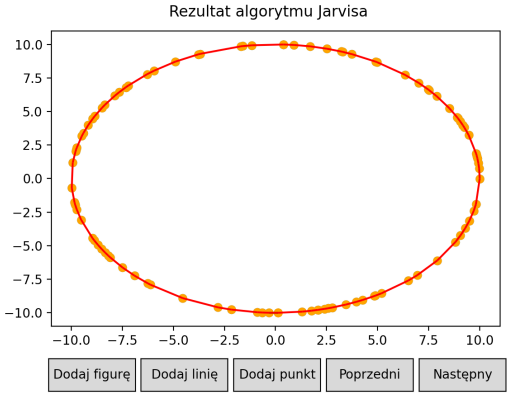
(c) Krok 3



(d) Krok 4



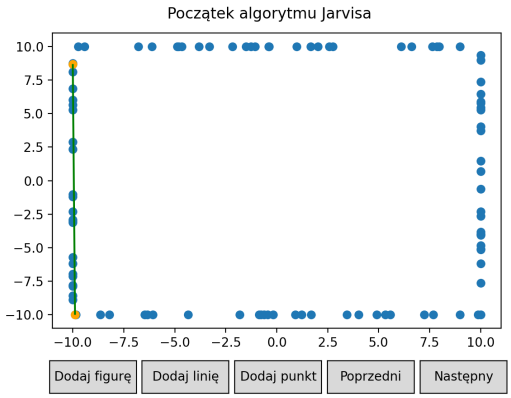
(e) Krok 5



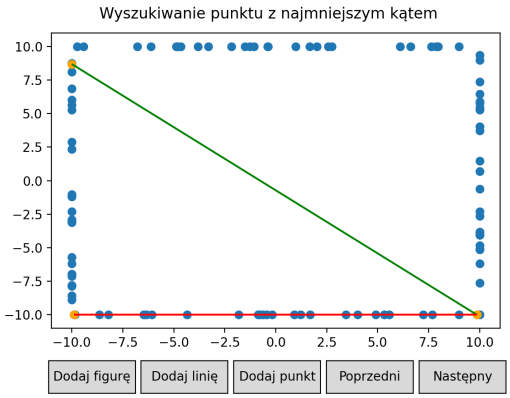
(f) Krok 6

Rysunek 10: Wizualizacja algorytmu Jarvisa dla zestawu danych B

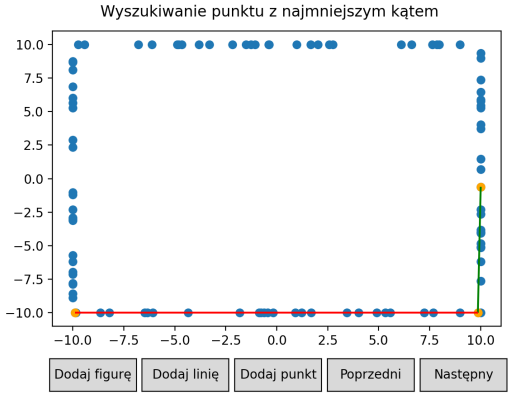
4.2.3 Zestaw danych C



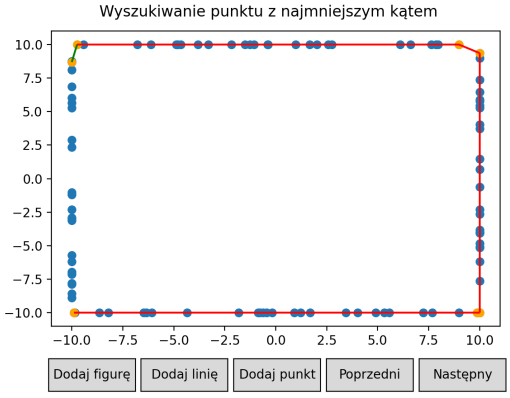
(a) Krok 1



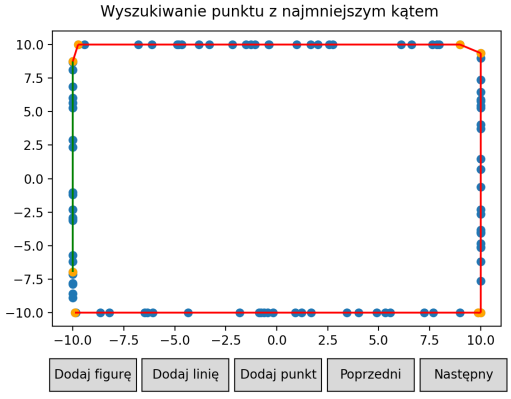
(b) Krok 2



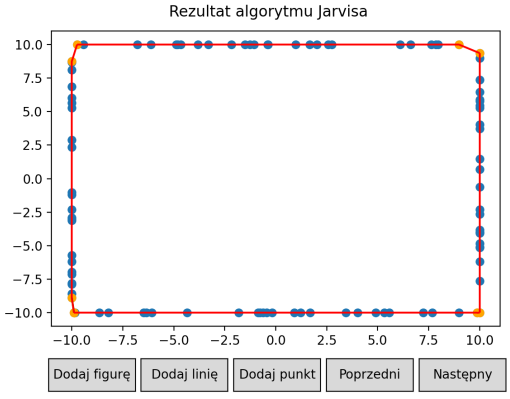
(c) Krok 3



(d) Krok 4



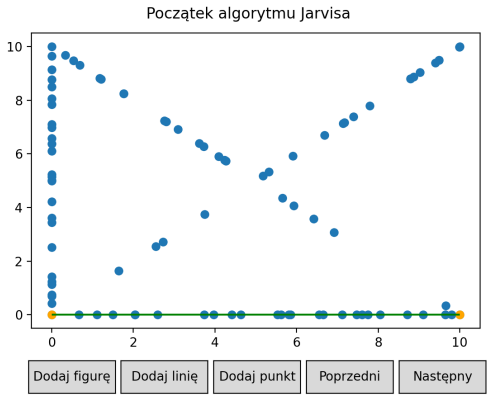
(e) Krok 5



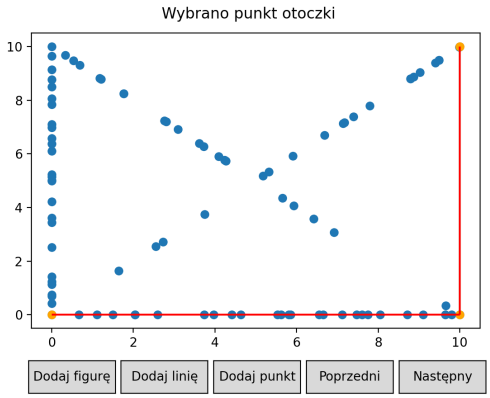
(f) Krok 6

Rysunek 11: Wizualizacja algorytmu Jarvisa dla zestawu danych C

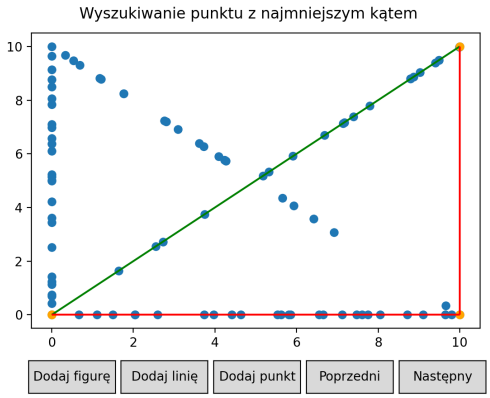
4.2.4 Zestaw danych D



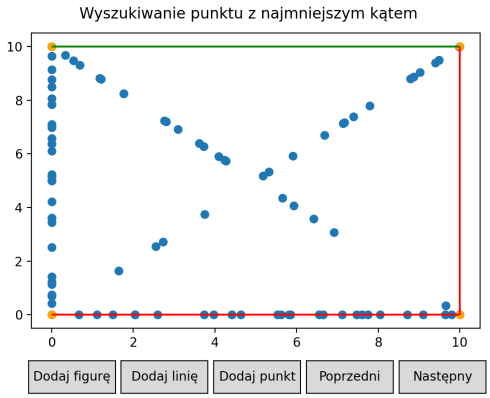
(a) Krok 1



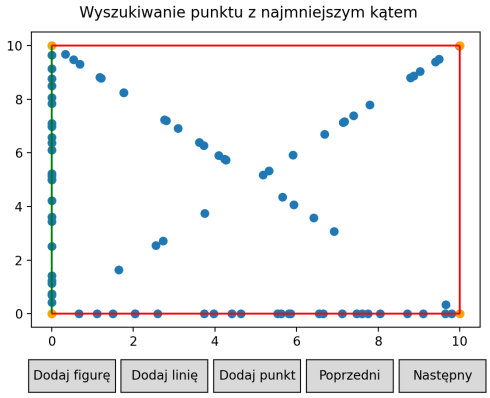
(b) Krok 2



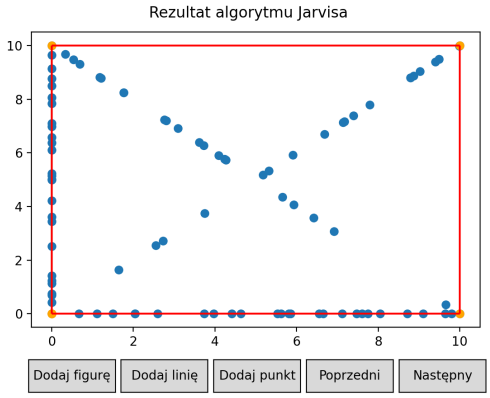
(c) Krok 3



(d) Krok 4



(e) Krok 5



(f) Krok 6

Rysunek 12: Wizualizacja algorytmu Jarvisa dla zestawu danych D

Analizując powyższe obrazy, łatwo dojść do wniosku, że wygenerowane zbiory danych zostały specjalnie dobrane, aby przetestować przypadki brzegowe algorytmów otoczkowych. Zbiory C oraz D testują algorytmy, gdy mamy do czynienia z większą liczbą punktów współliniowych, a zestaw B sprawdza, czy na pewno algorytm zakwalifikuje wszystkie punkty okręgu. Odpowiednie modyfikacje opisane w rozdziale 1. pozwalają na sprawne radzenie sobie z takimi przeszkodami.

5 Porównanie czasów działania algorytmów

W celu zmierzenia czasów działania algorytmów skorzystano z funkcji `process_time`, będącej częścią biblioteki `time`. Funkcja ta, w porównaniu do innych funkcji z tej biblioteki, oblicza czas działania procesora dla danego procesu (system and user CPU time), nie bierze pod uwagę np. czasu, gdy program korzysta z funkcji `sleep`.

Dla uniknięcia błędów związanych z precyzją obliczeń, dla zestawów danych zwiększono zakresy losowanych punktów czy też wielkości wielokątów.

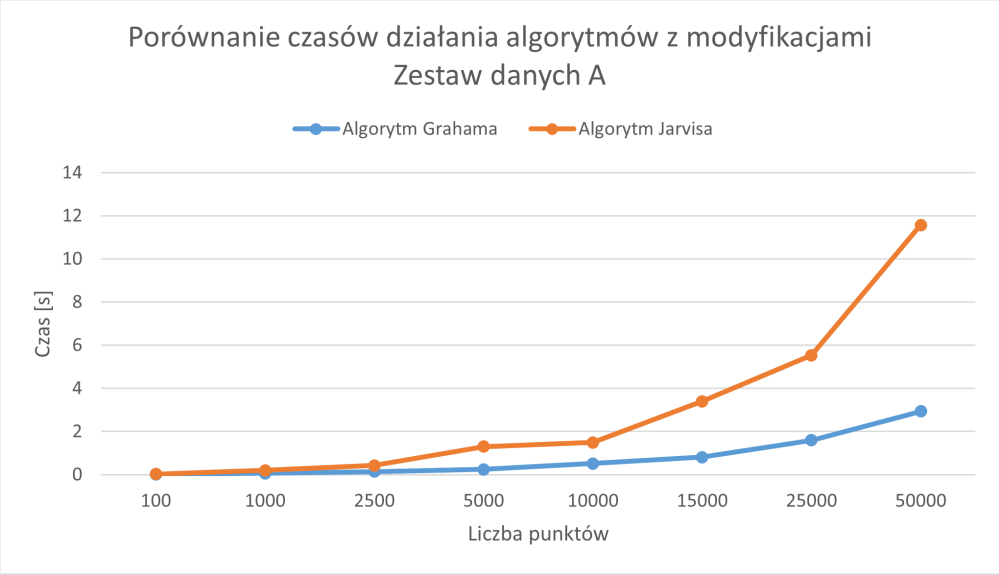
Każda zaimplementowana funkcja danego algorytmu posiada opcje dodawania scen, w celu wizualizacji działania. Jednakże dodatkowe modyfikacje mogą znacznie zwolnić algorytm. Została zaimplementowana osobna funkcja dla algorytmów bez możliwości m.in. dodawania scen, aby obliczyć tylko czas algorytmu. Dla każdego zestawu danych są podane dwie tabele: czasy działania dla algorytmów z opcjami m.in. tworzenie scen oraz czasy dla samych algorytmów (dalej: algorytmy z modyfikacjami) oraz algorytmy bez dodatkowych opcji.

5.1 Zestaw danych A

Dla zestawu danych A, to algorytm Grahama klasyfikował punkty szybciej, co było spowodowane złożonością tego algorytmu. Dla takiego zestawu danych otoczka może być duża, przez co algorytm Jarvisa może, w skrajnym przypadku, osiągnąć złożoność $O(n^2)$.

Przedział losowania	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$[-100, 100]$	100	0.01563	0.03125
$[-300, 300]$	1000	0.06250	0.20313
	2500	0.14063	0.42188
	5000	0.25000	1.29688
	10000	0.51563	1.50000
	15000	0.81250	3.40625
	25000	1.59375	5.53125
	50000	2.93750	11.57813

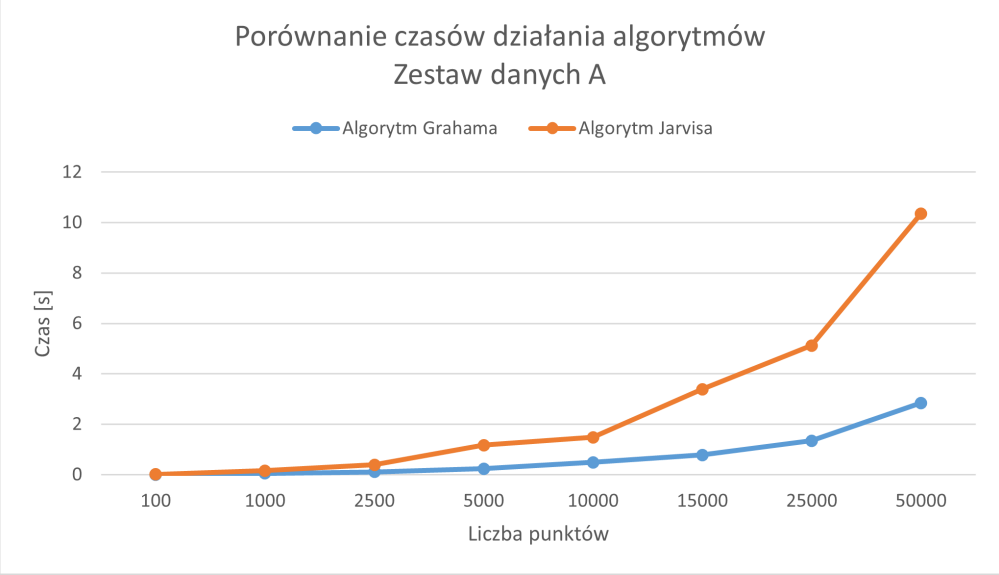
Tabela 1: Tabela czasów działania algorytmów z modyfikacjami dla zestawu danych A



Rysunek 13: Wykres czasów działania algorytmów z modyfikacjami dla zestawu danych A

Przedział losowania	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$[-100, 100]$	100	0.00000	0.01563
$[-300, 300]$	1000	0.04688	0.15625
	2500	0.10938	0.39063
	5000	0.23438	1.17188
	10000	0.48438	1.48438
	15000	0.78125	3.39063
	25000	1.34375	5.12500
	50000	2.84375	10.35938

Tabela 2: Tabela czasów działania algorytmów dla zestawu danych A



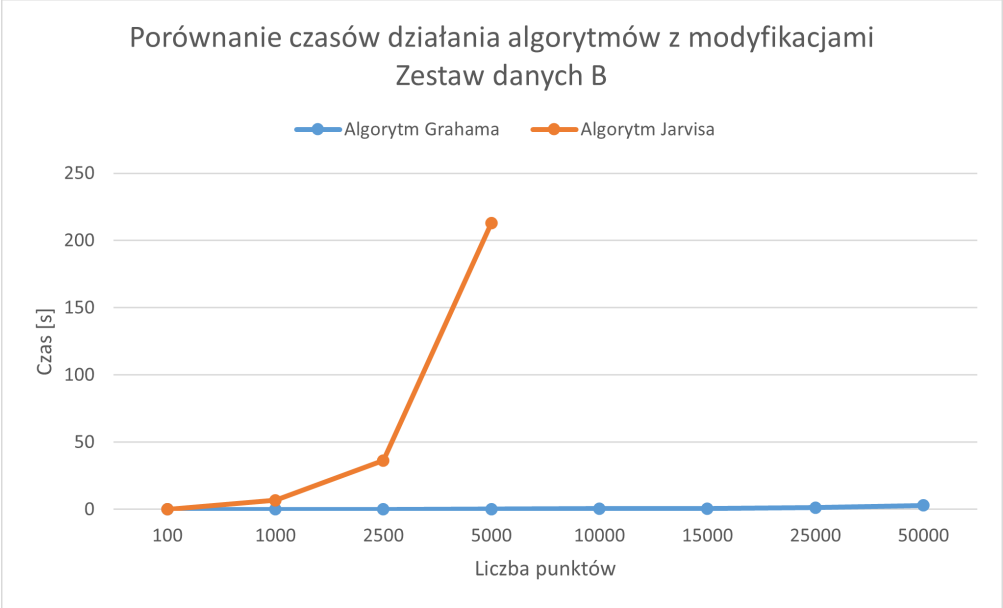
Rysunek 14: Wykres czasów działania algorytmów dla zestawu danych A

5.2 Zestaw danych B

Zestaw danych B to okrąg, zatem otoczka wypukła to właśnie wszystkie punkty z tego zbioru. Nie są zaskoczeniem duże czasy działania algorytmu Jarvisa w porównaniu do algorytmu Grahama, który, zgodnie z oczekiwaniami, działał szybciej.

Środek i promień okręgu	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$S(0,0),\ R = 100$	100	0.01563	0.10938
$S(0,0),\ R = 300$	1000	0.06250	6.90625
	2500	0.12500	36.46875
	5000	0.26563	212.85938
	10000	0.62500	-
	15000	0.68750	-
	25000	1.37500	-
	50000	2.98438	-

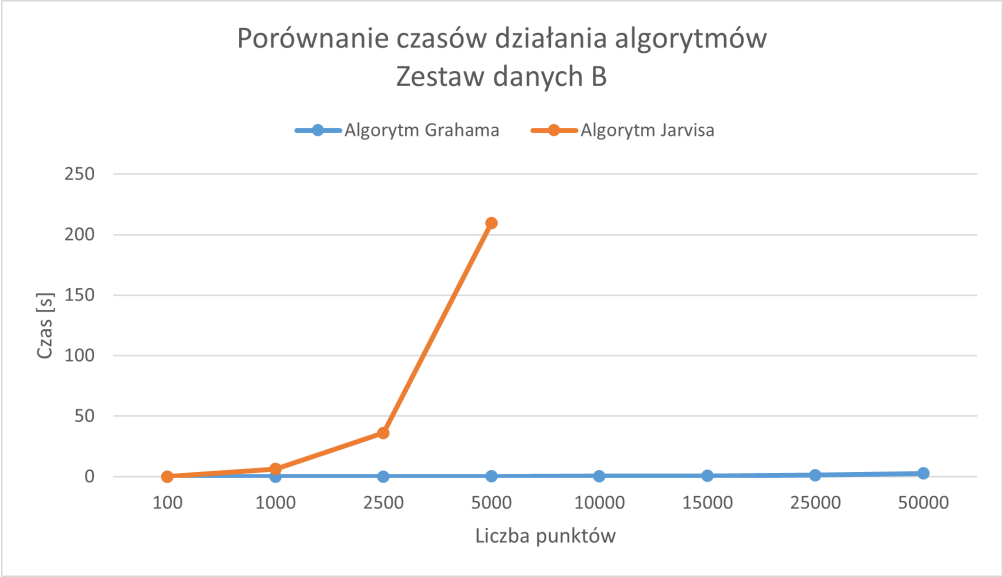
Tabela 3: Tabela czasów działania algorytmów z modyfikacjami dla zestawu danych B



Rysunek 15: Wykres czasów działania algorytmów z modyfikacjami dla zestawu danych B

Środek i promień okręgu	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$S(0,0),\ R=100$	100	0.00000	0.09375
$S(0,0),\ R=300$	1000	0.04688	6.29688
	2500	0.10938	36.00000
	5000	0.20313	209.54688
	10000	0.43750	-
	15000	0.67188	-
	25000	1.25000	-
	50000	2.64063	-

Tabela 4: Tabela czasów działania algorytmów dla zestawu danych B



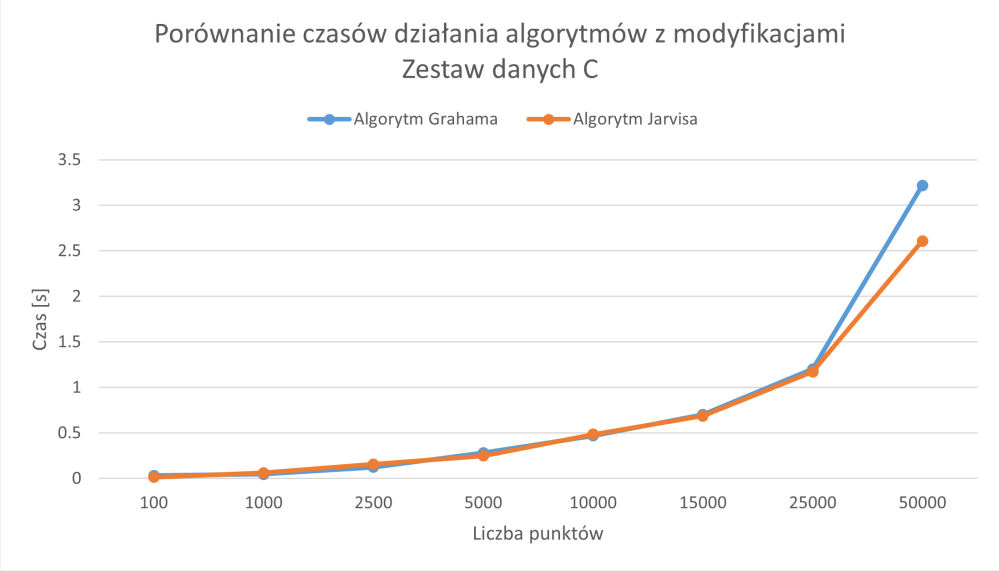
Rysunek 16: Wykres czasów działania algorytmów dla zestawu danych B

5.3 Zestaw danych C

Ten zestaw danych miał za zadanie przetestować czy algorytm potrafi klasyfikować punkty współliniowe. Ponieważ mamy do czynienia z prostokątem, liczba punktów w otoczce będzie stosunkowo mała, w najlepszym przypadku równa cztery. Oba algorytmy przyjmowały porównywalne czasy wykonywania programu dla mniejszych liczb punktów, dla większej liczby punktów lepiej działał algorytm Jarvisa.

Wierzchołki prostokąta	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$(-10,-10),(10,-10),(10,10),(-10,10)$	100	0.03125	0.01563
$(0,0),(30,0),(30,40),(0,40)$	1000	0.04688	0.06250
	2500	0.12500	0.15625
	5000	0.28125	0.25000
	10000	0.46875	0.48438
	15000	0.70313	0.68750
	25000	1.20313	1.17188
	50000	3.21875	2.60938

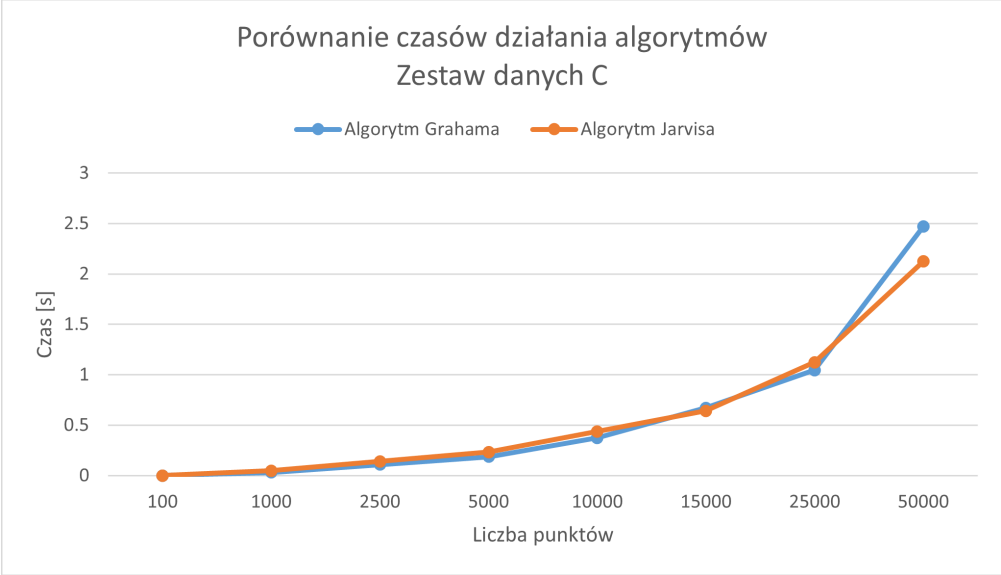
Tabela 5: Tabela czasów działania algorytmów z modyfikacjami dla zestawu danych C



Rysunek 17: Wykres czasów działania algorytmów z modyfikacjami dla zestawu danych C

Wierzchołki prostokąta	Liczba punktów	Czasy działania algorytmu $[s]$	
		Grahama	Jarvisa
$(-10,-10),(10,-10),(10,10),(-10,10)$	100	0.00000	0.00000
$(0,0),(30,0),(30,40),(0,40)$	1000	0.03125	0.04688
	2500	0.10938	0.14063
	5000	0.18750	0.23438
	10000	0.37500	0.43750
	15000	0.67188	0.64063
	25000	1.04688	1.12500
	50000	2.46875	2.12500

Tabela 6: Tabela czasów działania algorytmów dla zestawu danych C



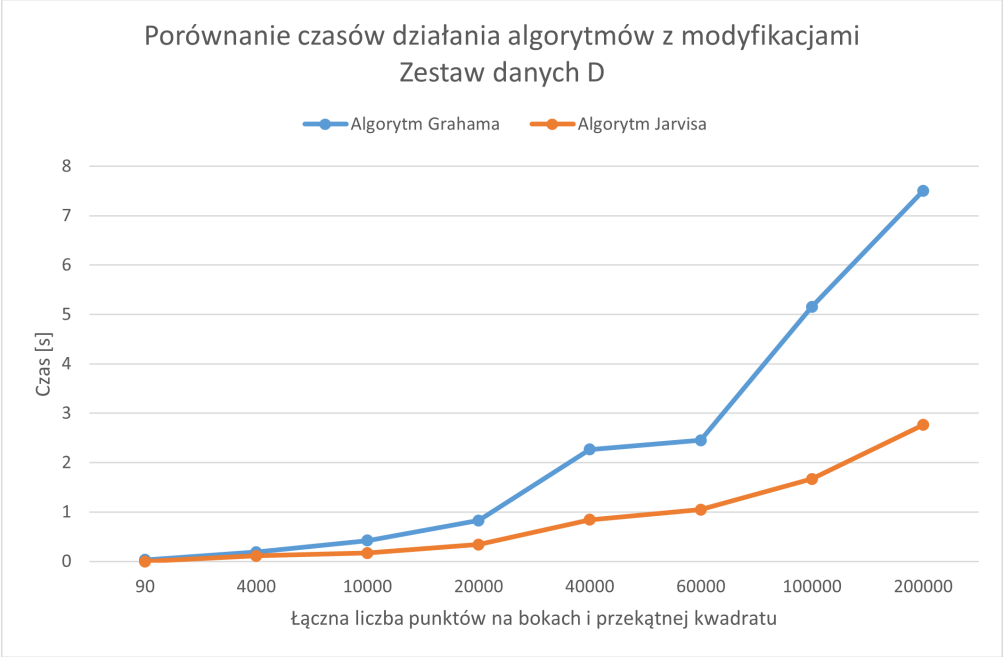
Rysunek 18: Wykres czasów działania algorytmów dla zestawu danych C

5.4 Zestaw danych D

W tym zestawie lepiej poradził sobie algorytm Jarvisa, ponieważ liczba punktów należących do otoczki będzie zawsze równa cztery, a ponieważ złożoność algorytmu to $O(nk)$, gdzie k, to liczba punktów w otoczce wypukłej, to algorytm klasyfikował punkty w złożoności liniowej, a algorytm Grahama ze złożonością $O(n \log n)$.

Wierzchołki kwadratu	Liczba punktów		Czasy działania algorytmu $[s]$	
	na przekątnej	na bokach	Grahama	Jarvisa
$(-10,-10),(10,-10),(10,10),(-10,10)$	20	25	0.03125	0.00000
$(0,0),(30,0),(30,30),(0,30)$	1000	1000	0.18750	0.10938
	2500	2500	0.42188	0.17188
	5000	100	0.82813	0.34375
	10000	100	2.26563	0.84375
	15000	100	2.45313	1.04688
	25000	100	5.15625	1.67188
	50000	100	7.50000	2.76563

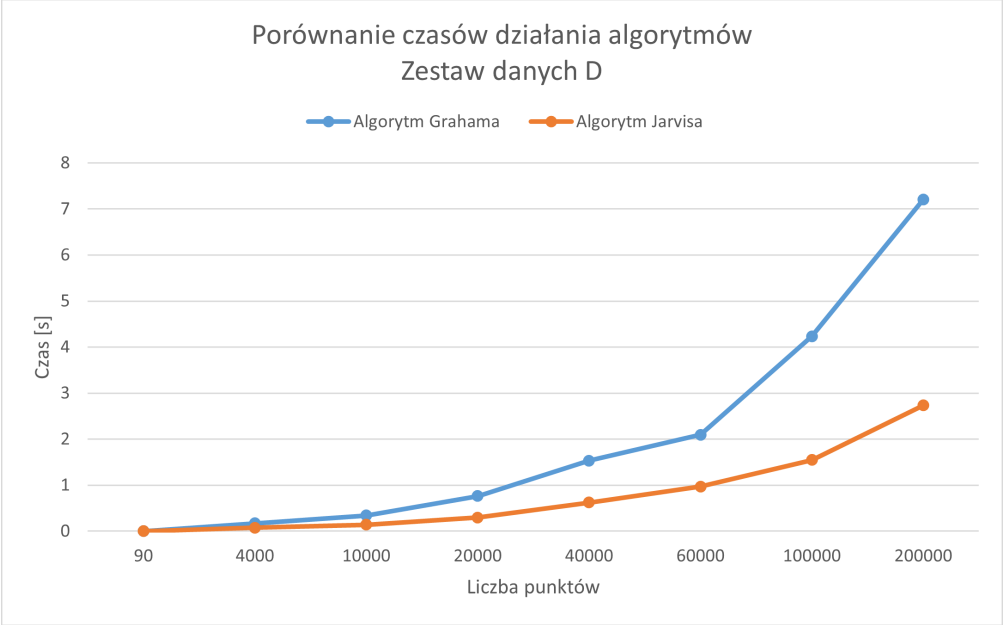
Tabela 7: Tabela czasów działania algorytmów z modyfikacjami dla zestawu danych D



Rysunek 19: Wykres czasów działania algorytmów z modyfikacjami dla zestawu danych D

Wierzchołki kwadratu	Liczba punktów		Czasy działania algorytmu $[s]$	
	na przekątnej	na bokach	Grahama	Jarvisa
$(-10,-10),(10,-10),(10,10),(-10,10)$	20	25	0.00000	0.00000
$(0,0),(30,0),(30,30),(0,30)$	1000	1000	0.17188	0.07813
	2500	2500	0.34375	0.14063
	5000	100	0.76563	0.29688
	10000	100	1.53125	0.62500
	15000	100	2.09375	0.96875
	25000	100	4.23438	1.54688
	50000	100	7.20313	2.73438

Tabela 8: Tabela czasów działania algorytmów dla zestawu danych D



Rysunek 20: Wykres czasów działania algorytmów dla zestawu danych D

6 Wnioski

Po przeanalizowaniu czasów działania algorytmów dla czterech zestawów danych, bardzo trudno jest wybrać najlepszy algorytm do wyznaczania otoczki wypukłej. Dla zestawów danych A, B najlepszym wyborem okazał się algorytm Grahama, lecz dla zestawu danych C (dla większej liczby punktów) oraz D lepszym wyborem byłby algorytm Jarvisa. W teorii, gdybyśmy znali ograniczenie górne na liczbę punktów w otoczce, moglibyśmy odpowiednio wybierać algorytmy do danego zbioru danych. W praktyce jest to możliwe, gdybyśmy po prostu wygenerowali kilkaset zbiorów danych, znaleźli otoczki wypukłe i określili ograniczenia, lecz ten sposób raczej nie jest praktyczny. Zatem, gdy nie znamy ograniczeń na liczbę punktów w otoczce, lepiej wybrać algorytm Grahama, który powinien wyznaczyć otoczkę wypukłą w rozsądnym czasie.