Algorytmy Geometryczne

Sprawozdanie z ćwiczenia 2. „Otoczka wypukła”

Maciej Wiśniewski
Grupa 3 Poniedziałek 16.45 A

Data wykonania 13.11.2024 Data oddania 17.11.2024

1. Dane techniczne

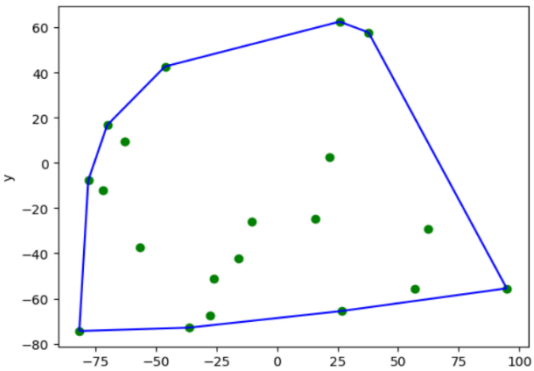
Specyfikacja komputera: system ***Ubuntu 24.04.01 Linux 5.15 x64***, procesor ***AMD Ryzen 7 5825U with Radeon 2GHz 8 rdzeni, 16GB pamięci RAM.***
Ćwiczenie zostało napisane w języku ***Python 3.9.20*** **w** ***Jupyter Notebook*** w środowisku programistycznym ***Visual Studio Code.*** Aby wykonać ćwiczenie posłużono się biblioteką ***numpy***. Do wykonania wizualizacji użyto narzędzia graficznego wykonanego przez ***Koło Naukowe BIT***.

2. Cel ćwiczenia

Celem ćwiczenia jest implementacja algorytmów ***Grahama*** i ***Jarvisa*** do wyznaczania **otoczki wypukłej**, porównanie ich złożoności czasowej oraz zastosowanie ich do obliczenia otoczek wypukłych dla wybranych zbiorów punktów. Zadanie obejmuje też wizualizację i opracowanie wyników oraz napisanie wniosków.

3. Wstęp teoretyczny

Podzbiór płaszczyzny Q nazywamy wypukłym, jeśli dla dowolnej pary punktów **p,q∈Q** odcinek pq jest całkowicie zawarty w **Q. Otoczka wypukła *CH(Q)*** zbioru Q to najmniejszy wypukły zbiór zawierający zbiór **Q**(Rysunek 1). W ramach tego laboratorium wyznaczę otoczki wypukłe przy użyciu algorytmów ***Grahama*** i ***Jarvisa***.



Rysunek 1 Przykładowa wizualizacja otoczki

Algorytm Grahama

Algorytm Grahama tworzy otoczkę wypukłą, wykorzystując **stos S**, na którym gromadzone są punkty mogące stanowić część otoczki wypukłej. Kolejne punkty ze zbioru **Q** są dodawane na stos, ale jeśli dany punkt nie należy do otoczki ***CH(Q)*** zostaje usunięty. Po zakończeniu działania algorytmu stos **S** zawiera wyłącznie punkty otoczki wypukłej ***CH(Q),*** uporządkowane przeciwnie do ruchu wskazówek zegara.

Zasada działania algorytmu jest następująca:

1. Znajdź punkt p_0 o najmniejszym położeniu y, jeśli jest kilka wybierz ten z najmniejszym x.
2. Posortuj pozostałe punkty względem p_0 w kolejności przeciwnie do ruchu wskazówek zegara, tworząc ciąg $\langle p_1, p_2, \ldots, p_m \rangle$ (jeśli kilka punktów tworzy ten sam kąt, wybierz najdalszy).
3. Utwórz pusty stos **S** i wstaw na niego punkty p_0, p_1, p_2.
4. Dla kolejnych punktów $\langle p_3, \ldots, p_m \rangle$**:** dopóki punkty S_{-2}, S_{-1}, p_i (punkt pod wierzchołkiem stosu, wierzchołek stosu oraz aktualnie badany punkt) tworzą tzw. prawostronny skręt(trójkąt S_{-2}, S_{-1}, p_i jest zorientowany zgodnie z ruchem wskazówek zegara), zdejmij wierzchołek stosu; następnie dodaj na stos punkt p_i.
5. Zwróć stos S - zawiera on punkty tworzące otoczkę wypukłą.

Szczegóły implementacji użytej w tutejszym przypadku.

1. Znajdujemy punkt o najmniejszym y, jeśli jest kilka to bierzemy pod uwagę najmniejszy x. Do tego użyto funkcji wbudowanej ***min()*** oraz warunku z funkcją anonimową.
2. Wybieramy nasz punkt referencyjny p_0 (otrzymany w poprzednim kroku) i sortujemy resztę punktów względem kąta, pod jakim punkty znajdują się względem naszego punktu referencyjnego oraz względem odległości punktów od punktu referencyjnego. W tej operacji kąt liczymy używając funkcji ***numpy.arctan2(),*** użyto tej funkcji trygonometrycznej, ponieważ jest łatwa i czytelna w implementacji, zwraca pełny zakres kątów (z zakresu [-π, π]), naturalnie rozróżnia kierunek obrotu, unika obliczania pierwiastka kwadratowego, jednakże wymaga uwagi, gdy różnica współrzędnych y-kowych wynosi 0. Wady innych funkcji: ***arcsin()*** – daje kąty tylko z zakresu [-π/2, π/2], wymaga liczenia pierwiastka, ***arccos()*** – daje kąty z zakresu [0, π], ***arcctg***() – daje kąty tylko z zakresu [-π/2, π/2].
3. Usuwamy punktu współliniowe poprzez porównywanie ich kąta – punkty współliniowe mają ten sam kąt.

5. Utwórz pusty stos **S** i wstaw na niego punkty p_0, p_1, p_2. Dla kolejnych punktów $\langle p_3,\ldots,p_m\rangle$: dopóki punkty S_{-2}, S_{-1}, p_i (punkt pod wierzchołkiem stosu, wierzchołek stosu oraz aktualnie badany punkt) tworzą tzw. prawostronny skręt(trójkąt S_{-2}, S_{-1}, p_i jest zorientowany zgodnie z ruchem wskazówek zegara), zdejmij wierzchołek stosu; następnie dodaj na stos punkt p_i. Tutaj używamy metody z użyciem wyznacznika macierzy ***mat_det_3x3*** aby wyznaczyć położenie punktu**.**
6. Zwróć stos ***S*** - zawiera on punkty tworzące otoczkę wypukłą.

Użyto tutaj funkcji trygonometrycznych, ponieważ okazały się bardziej wydajniejsze na testach niż metoda używająca wyznacznika.

Algorytm Jarvisa

Algorytm Jarvisa wyznacza otoczkę wypukłą dla zbioru punktów ***Q*** przy użyciu techniki nazywanej „owijaniem paczki" (ang. gift wrapping). Algorytm tworzy sekwencję $H = \langle p_1, p_2,\ldots, p_m\rangle$, która zawiera wierzchołki ***CH(Q)*** - otoczki wypukłej, uporządkowane przeciwnie do ruchu wskazówek zegara.

Zasada działania:

1. Znajdź punkt p_0, który ma najmniejszą współrzędną ***y*** (a jeśli jest kilka takich punktów, wybierz ten o najmniejszej współrzędnej ***x***).
2. Dodaj p_0 do sekwencji ***H***.
3. Każdy kolejny punkt p_i, dodawany do sekwencji, znajduje się jako punkt tworzący najmniejszy kąt względem poprzedniego punktu w sekwencji (w przypadku kilku punktów o tym samym kącie, wybierany jest ten najbardziej oddalony).
4. Algorytm kończy działanie, gdy następny znaleziony punkt jest równy punktowi p_0. Utworzona sekwencja ***H*** zawiera wierzchołki otoczki wypukłej.

Szczegóły implementacji użytej w tutejszym przypadku.

1. Znajdujemy punkt p_0, który ma najmniejszą współrzędną ***y*** (a jeśli jest kilka takich punktów, wybieramy ten o najmniejszej współrzędnej ***x***). Do tego użyto funkcji wbudowanej ***min()*** oraz warunku z funkcją anonimową.
2. Dodajemy punkt do sekwencji.
3. Każdy kolejny punkt p_i znajduje się jako punkt tworzący najmniejszy kąt względem poprzedniego punktu w sekwencji (w przypadku kilku punktów o tym samym kącie, wybierany jest ten najbardziej oddalony). Tutaj używamy metody z użyciem wyznacznika macierzy ***mat_det_3x3*** aby wyznaczyć położenie punktu**.**
4. Algorytm kończy działanie, gdy następny znaleziony punkt jest równy punktowi p_0. Utworzona sekwencja ***H*** zawiera wierzchołki otoczki wypukłej.

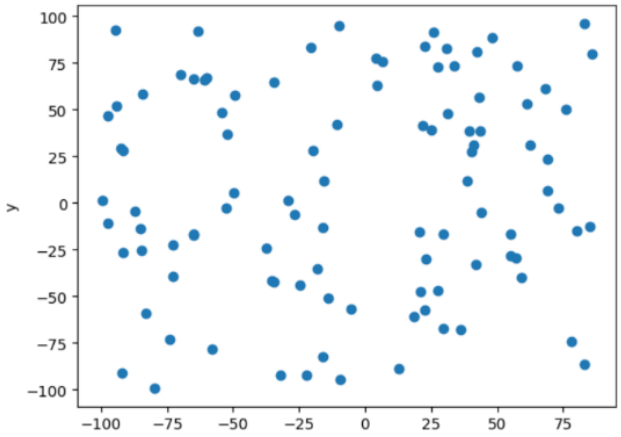
Użyto tutaj wyznacznika macierzy, ponieważ okazał się bardziej wydajniejszy na testach niż funkcje trygonometryczne.

Szczegółowy opis otoczki wypukłej, jej zastosowań oraz algorytmów znajduje się w pliku implementacyjnym.

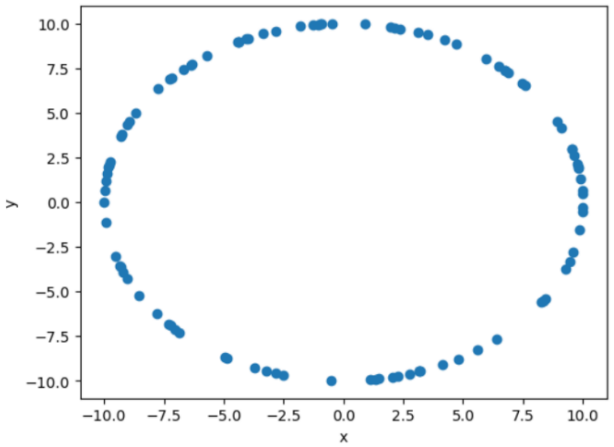
4. Realizacja ćwiczenia

Początkowo, w celu realizacji ćwiczenia przygotowano następujące cztery zbiory punktów:

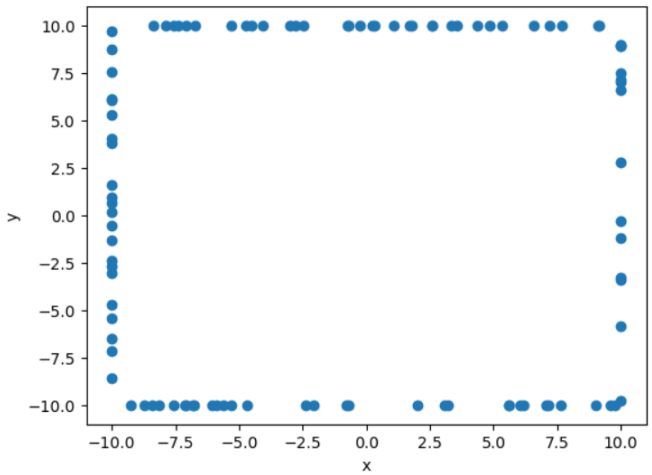
- **Zbiór A**: 100 losowo wygenerowanych punktów o współrzędnych w przedziale [-100,100] (Rysunek 2)
- **Zbiór B**: 100 losowo wygenerowanych punktów na okręgu o środku O (0,0) i promieniu R = 10 (Rysunek 3)
- **Zbiór C**: 100 losowo wygenerowanych punktów znajdujących się na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10) (Rysunek 4)
- **Zbiór D** zawierający wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10) oraz po 25 punktów na bokach leżących na osiach i po 20 punktów na przekątnych (Rysunek 5)



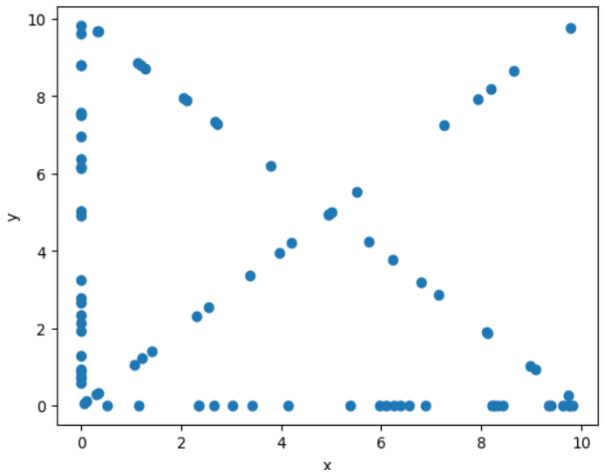
Rysunek 2 Wizualizacja ***Zbioru A***



Rysunek 3 Wizualizacja ***Zbioru B***



Rysunek 4 Wizualizacja ***Zbioru C***



Rysunek 5 Wizualizacja ***Zbioru D***

5. Testowe użycie algorytmów Grahama i Jarvisa

Dla każdego z powyższych zbiorów obliczono otoczkę z wykorzystaniem algorytmów ***Grahama*** i ***Jarvisa***.

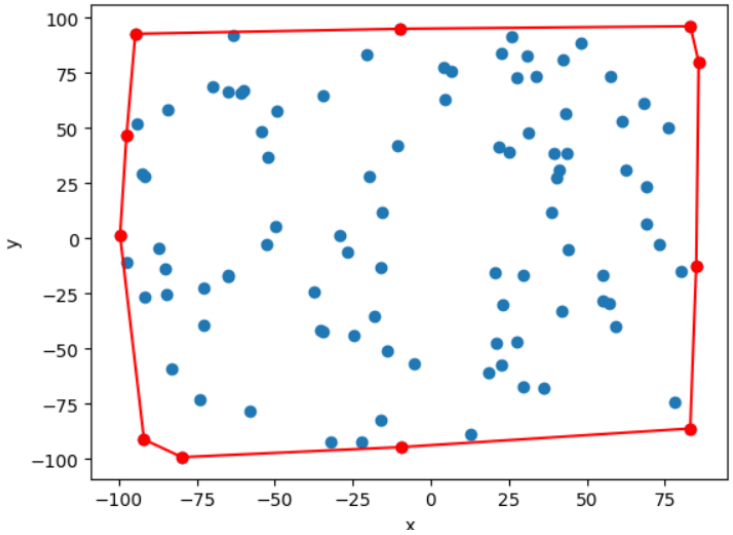
W poniższej tabeli przedstawiano liczbę wierzchołków otoczki wypukłej obliczonej przez każdy z dwóch algorytmów dla każdego ze zbiorów(Tabela 1).

Zbiór punktów	Liczba wierzchołków otoczki – algorytm Grahama	Liczba wierzchołków otoczki – algorytm Jarvisa
Zbiór A	11	11
Zbiór B	100	100
Zbiór C	8	8
Zbiór D	4	4

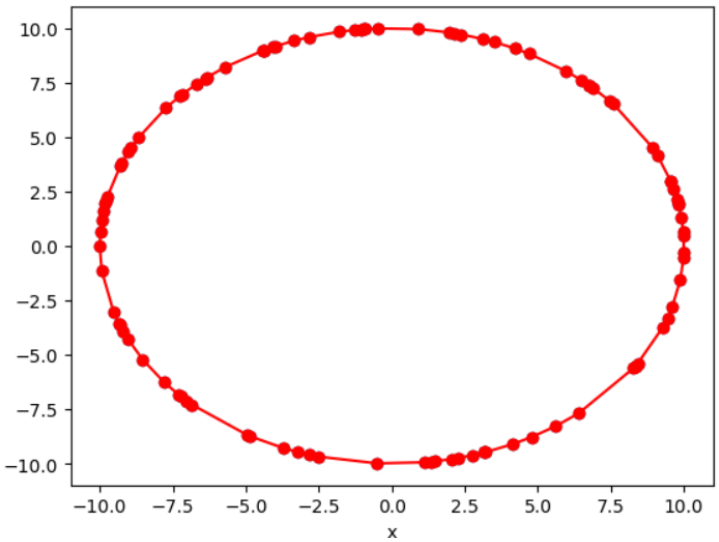
Tabela 1 Porównanie wyników algorytmu ***Grahama*** i ***Jarvisa*** dla zbiorów A-D

Jak zostało napisane w Tabeli 1, oba algorytmy wskazują taką samą liczbę wierzchołków otoczki dla zbiorów A-D. Okazuje się, że istotnie są to te same otoczki, dlatego tutaj zostały umieszczone tylko pojedyncze

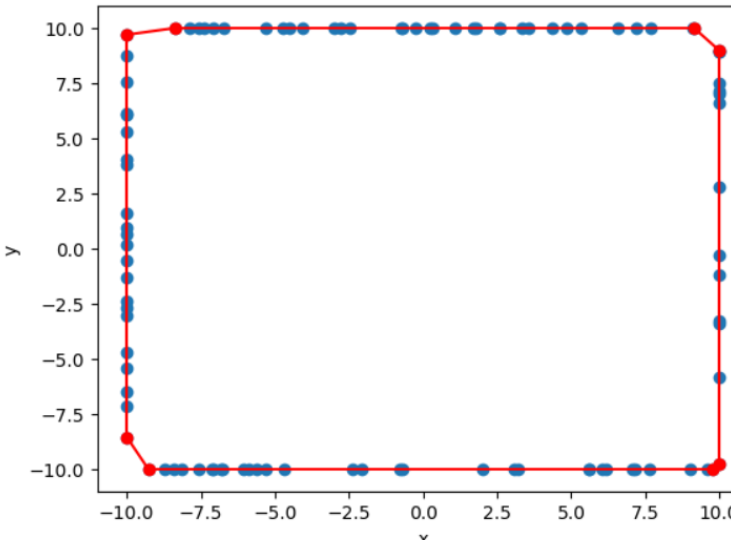
przykłady otoczek(Rysunek 6, 7, 8, 9). Wszystkie przykłady są dostępne w pliku z kodem. Na poniższych rysunkach zaznaczono na czerwono punkty należące do otoczki jak i samą otoczkę. W celu wyraźniejszego uchwycenie działalności algorytmów ***Grahama*** i ***Jarvisa*** wykonano wizualizację poszczególnych kroków działania algorytmów w postaci plików ***GIF***, gdzie na zielono oznaczano punkty w otoczce, na żółto krawędzie otoczki, a na czerwono punkty, które nie znajdą się w otoczce. Wszystkie pliki ***GIF*** są dostępne w oddzielnym pliku.



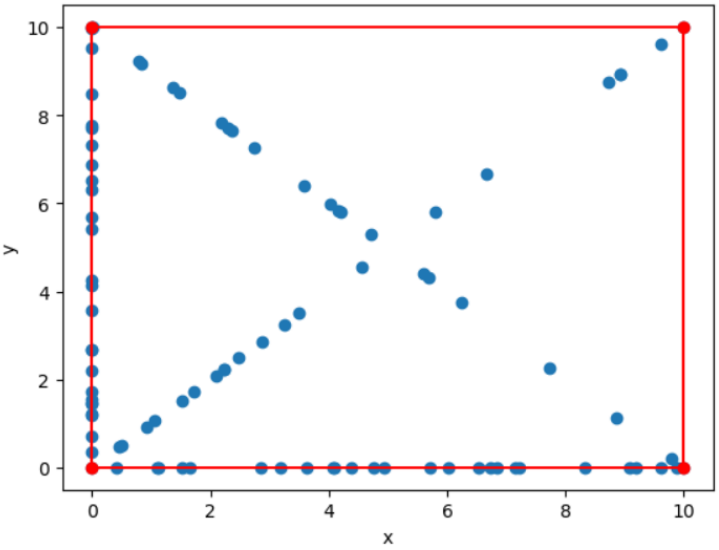
Rysunek 6 Otoczka otrzymana algorytmem ***Grahama*** i ***Jarvisa*** dla punktów ze ***Zbioru A***



Rysunek 7 Otoczka otrzymana algorytmem ***Grahama*** i ***Jarvisa*** dla punktów ze ***Zbioru B***



Rysunek 8 Otoczka otrzymana algorytmem ***Grahama*** i ***Jarvisa*** dla punktów ze ***Zbioru C***



Rysunek 9 Otoczka otrzymana algorytmem ***Grahama*** i ***Jarvisa*** dla punktów ze ***Zbioru D***

6. Porównanie złożoności czasowej algorytmów

Po przetestowaniu algorytmów ***Grahama*** i ***Jarvisa*** na ***Zbiorach A-D*** przystąpiono do porównania złożoności wydajnościowej algorytmów, a szczegółowiej złożoności czasowej. Teoretyczna złożoność czasowa algorytmu ***Grahama*** to $O(nlogn)$, natomiast algorytmu ***Jarvisa*** $O(kn)$, gdzie n oznacza liczbę wierzchołków zbioru, dla którego wyznaczamy otoczkę, a k to liczba wierzchołków otoczki. Pesymistycznie zatem algorytm ***Jarvisa*** ma złożoność $\Theta(n^2)$.

W celu porównania działalność obliczeniowej zostały utworzone nowe zbiory:

• ***Nowy Zbiór A***: n losowych punktów o współrzędnych z przedziału [-1000, 1000]

• ***Nowy Zbiór B***: m losowych punktów leżących na okręgu o środku w punkcie (100,100) i promieniu R=500

• ***Nowy Zbiór C***: n losowych punktów leżących na bokach prostokąta o wierzchołkach (-200,50), (-200,-150), (100,-150), (100,50)

• ***Nowy Zbiór D***: zawierający wierzchołki kwadratu (0,0), (100,0), (100,100), (0,100) oraz po $n1$ punktów leżących na osiach i po $n2$ punktów na przekątnych (łącznie n),

gdzie $n \in \{100,1000,5000,10000,50000\}$, $m \in \{10,100,500,1000,3000\}$ oraz $n1$ i $n2$ są równe odpowiednio $0{,}3 \cdot n - 1$ oraz $0{,}2 \cdot n - 1$.

Porównanie działalności algorytmów dla nowych zbiorów Tabel 2, Tabela 3.

Zbiór	Algorytm	Liczba punktów				
		100	1000	5000	10000	50000
Nowy Zbiór A	Graham	0.0005s	0.0043s	0.0153s	0.0279s	0.2103s
	Jarvis	0.0004s	0.0045s	0.0312s	0.0652s	0.3166s
Nowy Zbiór C	Graham	0.0004s	0.0030s	0.0152s	0.0320s	0.2048s
	Jarvis	0.0003s	0.0036s	0.0157s	0.0309s	0.1578s
Nowy Zbiór D	Graham	0.0003s	0.0021s	0.0112s	0.0232s	0.1391s
	Jarvis	0.0002s	0.0016s	0.0088s	0.0192s	0.0946s

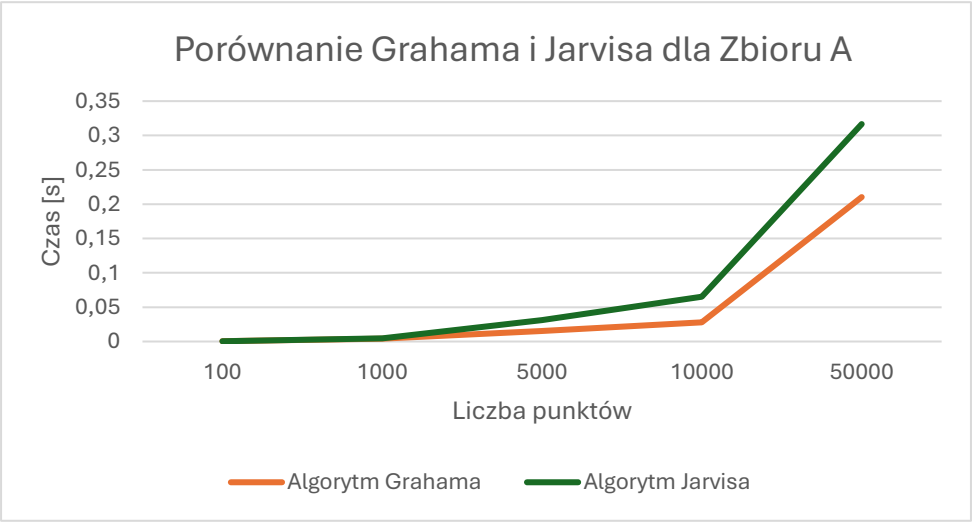
Tabela 2 Porównanie czasowe działania algorytmów ***Grahama*** i ***Jarvisa*** dla ***Nowych Zbiorów A,C.D***

Zbiór	Algorytm	Liczba punktów				
		10	100	500	1000	3000
Nowy Zbiór B	Graham	0.0001s	0.0003s	0.0016s	0.0032s	0.0103s
	Jarvis	0.0001s	0.0057s	0.1252s	0.5124s	4.5908s

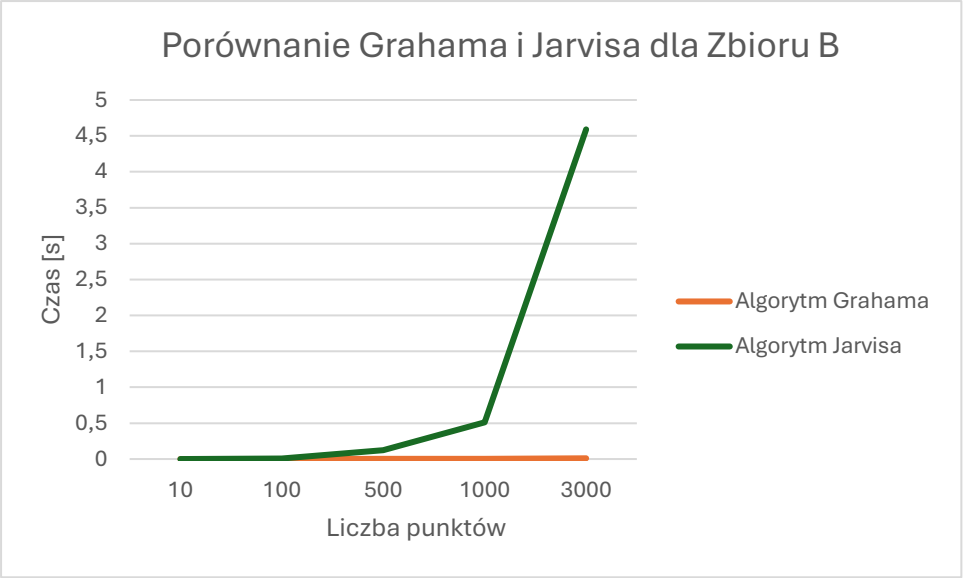
Tabela 3 Porównanie czasowe działania algorytmów ***Grahama*** i ***Jarvisa*** dla ***Nowego Zbioru B***

7. Przedstawienie danych na wykresach

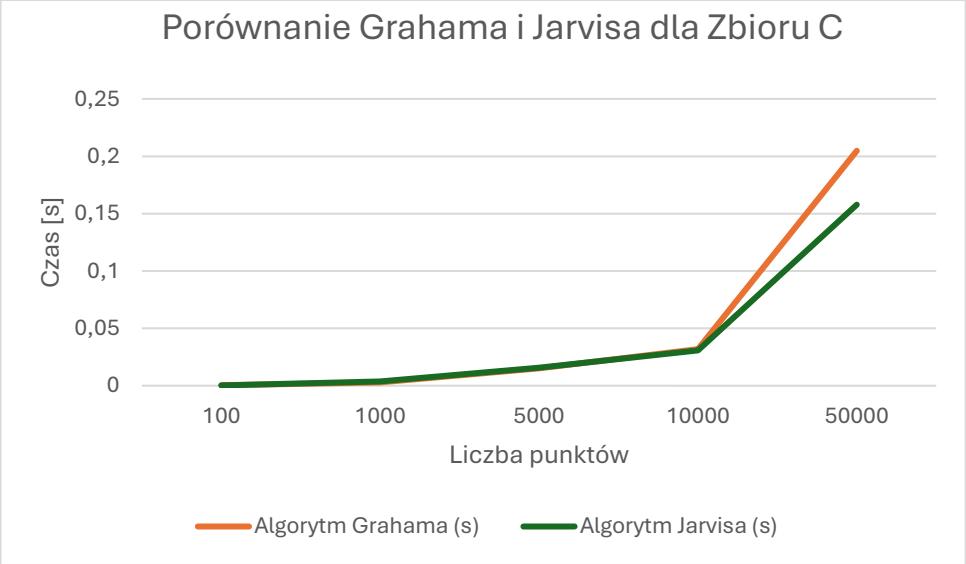
Na poniższych wykresach (Wykresy 1-4) zostały przedstawione liniowe zależności algorytmów wyznaczania otoczki dla wybranych zbiorów.



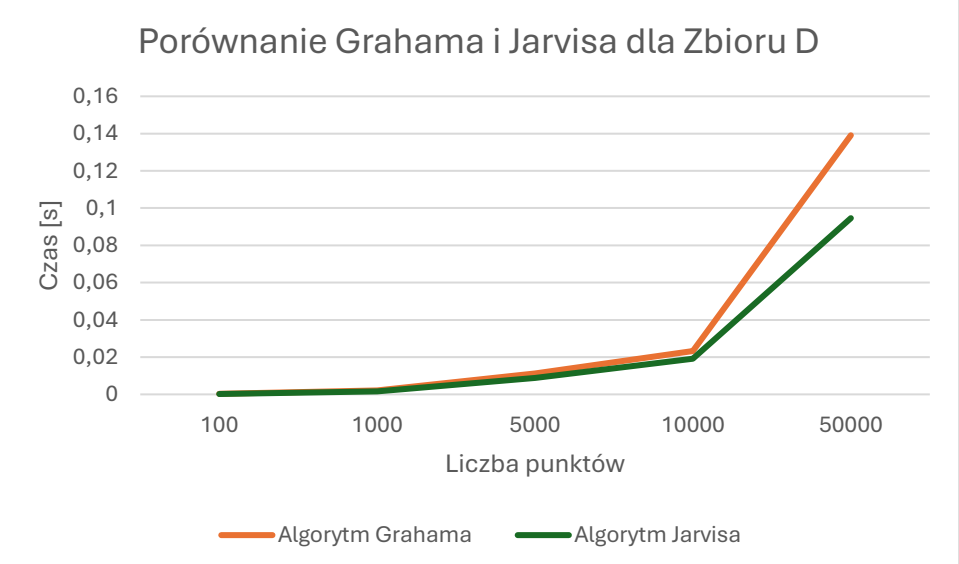
Wykres 1 Porównanie Grahama i Jarvisa dla Nowego Zbioru A



Wykres 2 Porównanie Grahama i Jarvisa dla Nowego Zbioru B



Wykres 3 Porównanie Grahama i Jarvisa dla Nowego Zbioru C



Wykres 4 Porównanie Grahama i Jarvisa dla Nowego Zbioru D

Jak pokazano na Wykresach(1-4), oba algorytmy działają podobnie dla małej liczby punktów(do 1000 dla zbiorów A, C, D i 100 dla B). Dla większych liczby punktów, dla zbiorów ***Nowy Zbiór A*** i ***Nowy Zbiór B*** algorytm ***Grahama*** wyliczał otoczkę o wiele szybciej niż algorytm ***Jarvisa***. Jest to spowodowane tym, że ***Nowy Zbiór A*** i ***Nowy Zbiór B*** posiadają sporo liczbę punktów w otoczce, więc rzeczywista różnica w długości czasowej wykonywania algorytmów pokrywa się z ich definicyjna złożonością czasową, ***Grahama*** to $O(nlogn)$, natomiast algorytmu ***Jarvisa*** $O(kn)$, gdzie n oznacza liczbę wierzchołków zbioru, dla którego wyznaczamy otoczkę, a k to liczba wierzchołków otoczki.

Dla zbiorów ***Nowy Zbiór C*** i ***Nowy Zbiór D***, przy większej liczbie punktów, algorytm ***Jarvisa*** okazał się szybszy od algorytmu ***Grahama.*** W zbiorach ***Nowy Zbiór C*** i ***Nowy Zbiór D*** liczba punktów w otoczce jest mniejsze od 9 (bardzo małe), więc tutaj również rzeczywista wartość czasu wykonywania algorytmów wskazuje na ich różnice w złożoności czasowej. Pokazuje to również, że stała przy złożoności czasowej algorytmu ***Jarvisa*** jest niewielki, co również wskazuje na to, że wykonana implementacja algorytmu ***Jarvisa*** jest poprawna.

Warto również zauważyć, że oś pozioma wykresu nie jest w skali liniowej, co oznacza, że większe nachylenie odcinków niekoniecznie świadczy o wyższym rzędzie lub większej stałej w złożoności. Punkty na wykresie zostały połączone, aby poprawić jego czytelność.

8. Wnioski

1. Porównanie algorytmów ***Grahama*** i ***Jarvisa***:
   - Wyniki algorytmów ***Grahama*** i ***Jarvisa*** dla wszystkich testowych zbiorów punktów (A-D) były zgodne, co potwierdza poprawność implementacji obu metod.
   - Algorytm ***Grahama*** działa szybciej niż ***Jarvisa*** dla dużych zbiorów, w których liczba punktów w otoczce wypukłej ***k*** jest znaczna w stosunku do liczby punktów całkowitych ***n***. Wynika to z niższej złożoności czasowej $O(nlogn)$ w porównaniu do pesymistycznej $\Theta(n^2)$ dla algorytmu ***Jarvisa***.
   - Algorytm ***Jarvisa*** działa szybciej niż ***Grahama*** w przypadku zbiorów, w których liczba punktów w otoczce ***k*** jest mała, np. w prostokątach lub kwadratach, co potwierdza jego złożoność $O(n\mathrm{k})$
2. Zastosowanie algorytmów:
   - Algorytm ***Grahama*** jest bardziej odpowiedni do przetwarzania dużych, losowych zbiorów punktów, gdzie $k \rightarrow n$, np. punktów generowanych losowo na płaszczyźnie.
3. Efektywność implementacji:
   - Wykorzystanie funkcji trygonometrycznych w algorytmie ***Grahama*** przyspieszyło sortowanie punktów względem kąta, co okazało się bardziej wydajne niż alternatywne metody.
   - Implementacja algorytmu ***Jarvisa*** z wykorzystaniem wyznacznika macierzy zapewniła szybkie i precyzyjne wyznaczanie relacji kątowych między punktami, co zredukowało koszty obliczeniowe w porównaniu do trygonometrycznych odpowiedników.
4. Wnioski z analizy czasowej:
   - Wyniki pomiarów czasowych dla obu algorytmów były zgodne z ich teoretyczną złożonością czasową.
   - Dla zbiorów punktów generowanych losowo (***Nowy Zbiór A, Nowy Zbiór B***) algorytm ***Grahama*** wyraźnie przewyższał **Jarvisa** przy większej liczbie punktów.
   - Dla regularnych struktur geometrycznych (***Nowy Zbiór C, Nowy Zbiór D***) algorytm ***Jarvisa*** okazał się efektywniejszy dzięki mniejszej liczbie punktów w otoczce wypukłej.
5. Podsumowanie praktycznych zastosowań:
   - Algorytm ***Grahama*** można zarekomendować do aplikacji wymagających szybkiego przetwarzania dużych i złożonych zbiorów danych.
   - Algorytm ***Jarvisa*** jest bardziej odpowiedni do problemów, gdzie otoczka wypukła składa się z małej liczby punktów, np. w analizie struktur geometrycznych lub w aplikacjach z ograniczonymi zasobami obliczeniowymi.