Algorytmy geometryczne
Sprawozdanie z Laboratorium 2

Adam Naumiec

naumiec@student.agh.edu.pl
410936
Grupa 4 – czwartek, 11:20-12:50, tydzień B

Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

Listopad MMXXII

Spis treści

1. Dane techniczne komputera, na którym wykonywano obliczenia

Wykorzystano:

- komputer z systemem $macOS\ 13\ Ventura$;
- czterordzeniowy procesor $Intel\ Core\ i5$;
- środowisko $Jupyter\ Notebook$, $JetBrains\ PyCharm$;
- język programowania $Python\ 3$ (3.10);
- wykorzystane biblioteki: $random$ (generowanie losowych zbiorów punktów), $math$ (funkcje trygonometryczne, π), $matplotlib$ (rysowanie wykresów), $time$ do sprawdzania czasu wykonania obliczeń;
- do przygotowania sprawozdania wykorzystano programy $Microsoft\ Word$, $Microsoft\ Excel$.

2. Cel ćwiczenia

W ćwiczeniu porównano działanie algorytmu Jarvisa i Grahama dla różnych losowo wygenerowanych zbiorów. Porównano między innymi czas działania dla zbiorów różnej specyfikacji (liczba punktów i sposób generowania). W ćwiczeniu również ukazano kolejne etapy działania algorytmów na wygenerowanych zbiorach.

3. Etapy realizacji ćwiczenia

Ćwiczenie wykonano realizując kolejne punkty:

1. generacja zbiorów losowych punktów,
2. wizualizacja zbiorów,
3. implementacja algorytmu Jarvisa i Grahama,
4. implementacja funkcji pomocniczych,
5. zastosowanie algorytmów do szukania otoczki wypukłej,
6. wizualizacja rezultatów działania algorytmów,
7. prezentacja kroków działania algorytmów,
8. testy wydajnościowe,
9. wnioski.

4. Generacja zbiorów losowych punktów

4.1. Zbiory

Wygenerowano 4 zbiory punktów:

1. **zbiór $\mathcal{A}$**: 100 *l*osowo wygenerowanych punktów o współrzędnych z przedziału $[-100, 100]$,
2. **zbiór $\mathcal{B}$**: 100 *l*osowych punktów leżących na okręgu o środku $(0, 0)$ i promieniu $R = 10$,
3. **zbiór $\mathcal{C}$**: 100 losowych punktów leżących na bokach prostokąta o wierzchołkach $(-10, 10), (-10, -10), (10, -10), (10, 10)$ oraz cztery punkty, po jednym na każdym wierzchołku,
4. **zbiór $\mathcal{D}$**: 25 losowych punktów na bokach i 20 na przekątnych kwadratu o wierzchołkach w punktach $(0, 0), (10, 0), (10, 10), (0, 10)$ oraz cztery punkty, po jednym na każdym wierzchołku

Punkty generowane były w przestrzeni $\mathbb{R}^2$ z metryką euklidesową (współrzędne punktów wyrażone były liczbami rzeczywistymi).

4.2. Generowanie losowych punktów i wykorzystanie funkcji bibliotecznych

Losowe liczby generowano za pomocą funkcji $uniform$ oraz $randint$ z biblioteki $random$. Funkcja ta generuje liczby pseudolosowe z podanego zakresu.

Zbiór $\mathcal{A}$ zawiera wygenerowane punkty zwarte na płaszczyźnie kartezjańskiej z punktami z zakresu $[-100, 100]$.

Do generacji punktów w zbiorze $\mathcal{B}$ wykorzystano z biblioteki $math$ funkcje sin do obliczania sinusa i cos do obliczania cosinusa. Funkcje te zwracają wartości funkcji trygonometrycznych dla podanych argumentów. Wykorzystanie funkcji trygonometrycznych do generowania punktów znajdujących się na okręgu pozwoliło na ich równomierne rozmieszczenie, ponieważ prędkość krzywej jest stała. Wykorzystano także biblioteczną wartość liczby π.

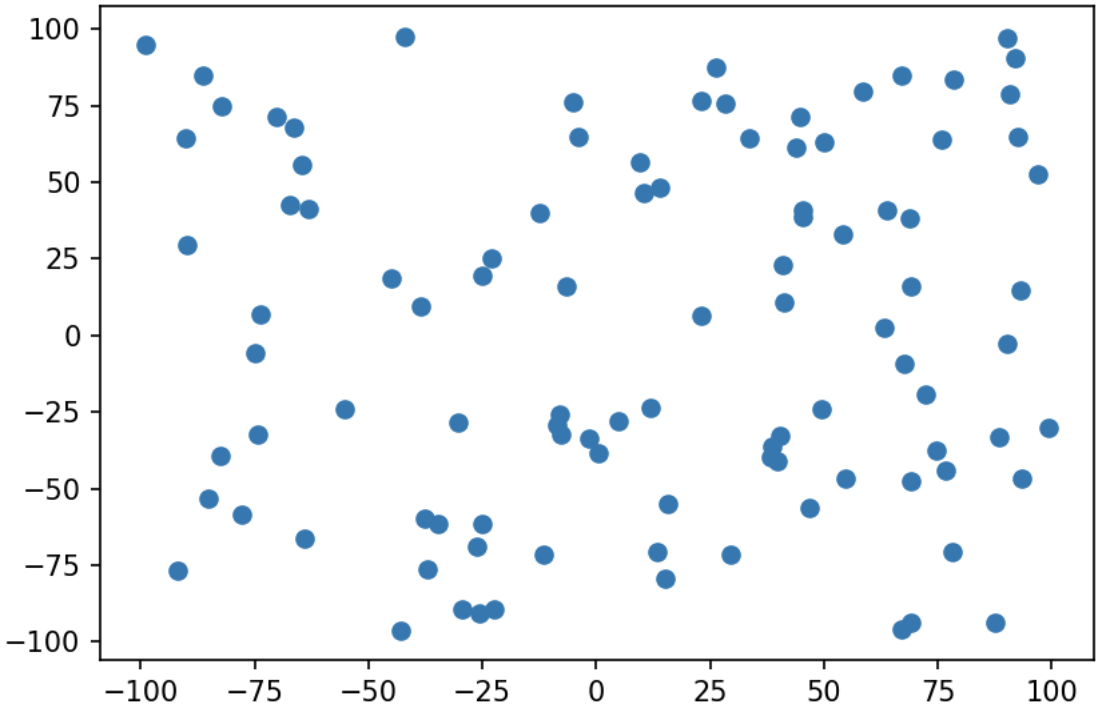
Zbiór $\mathcal{C}$ zawiera punkty rozmieszczone na bokach prostokąta oraz po jednym punkcie na każdym wierzchołku.

W zbiorze $\mathcal{D}$ generwano punkty leżące na bokach, które pokrywały się z osiami OX i OY, przekątnych kwadratu oraz po jednym punkcie na każdym wierzchołku.

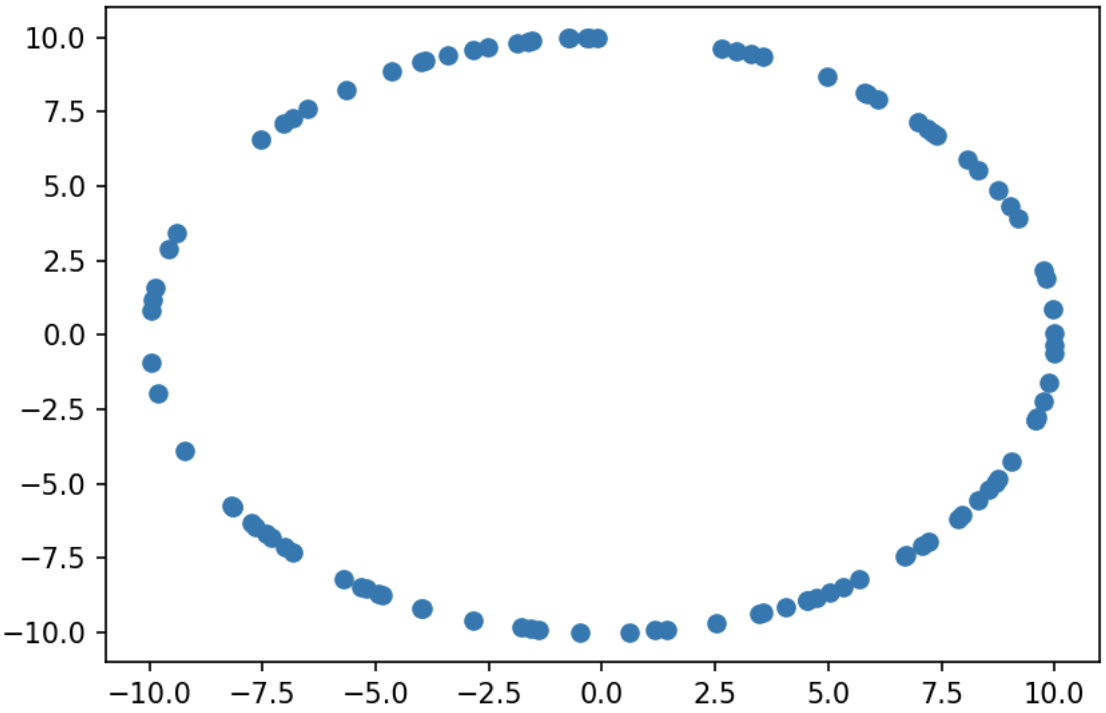
Do wizualizacji punktów na płaszczyźnie kartezjańskiej wykorzystano dostarczone na laboratoriach narzędzie graficzne korzystające z bibliotek

matplotlib oraz *numpy*. Pierwsza z nich pozwala na rysowanie wykresów, druga dostarcza wiele narzędzi przydatnych w algorytmice i obliczeniach komputerowych.

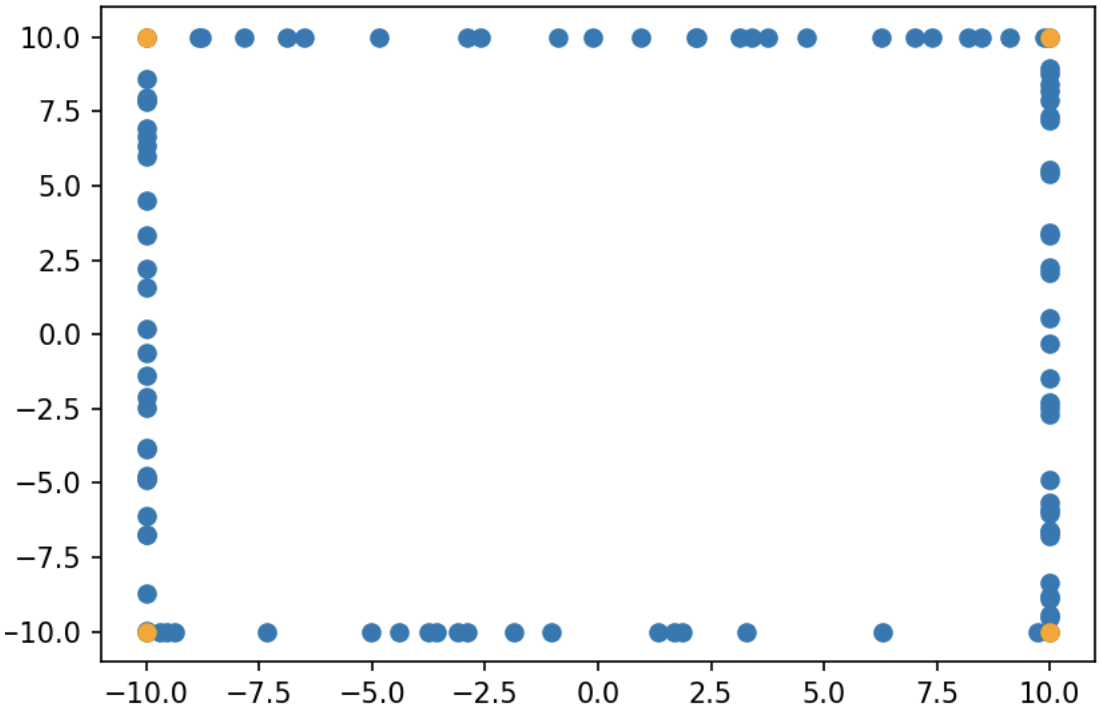
Wykorzystano także bibliotekę *time* do sprawdzenia czasu wykonania obliczeń. Przygotowano także funkcje wykorzystujące algorytmy do znajdowania otoczki wypukłej, ale nie generujące jej wizualizacji. Funkcje te użyto do sprawdzenia czasu wykonywania algorytmów Jarvisa i Grahama, ponieważ nie wykonywana jest w nich instrukcja warunkowa, która miałaby sprawdzać czy generować jej wizualizację. Sposób taki wybrano, by otrzymany czas odnosił się jedynie do wykonania właściwych instrukcji algorytmów znajdowania otoczki.



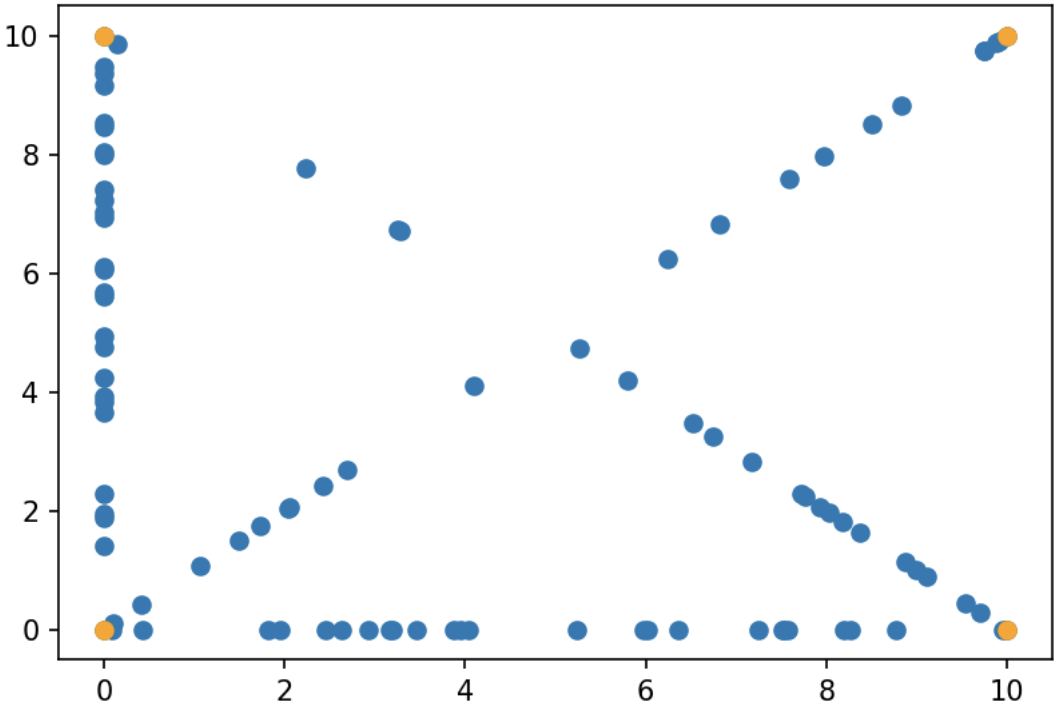
Rysunek 4.3 a) Zbiór $\mathcal{A}$



Rysunek 4.3 b) Zbiór $\mathcal{B}$



Rysunek 4.3 c) Zbiór $\mathcal{C}$
(na pomarańczowo zaznaczone punkty w wierzchołkach)



Rysunek 4.3 d) Zbiór $\mathcal{D}$
(na pomarańczowo zaznaczone punkty w wierzchołkach)

5. Algorytmy

5.1. Użyte algorytmy znajdowania otoczki wypukłej

Do obliczeń wykorzystano:

1. Algorytm Jarvisa – złożoność $O(nk)$ – potencjalnie $O(n^2)$,
2. Algorytm Grahama – złożoność $O(n \log n)$,

gdzie: n to liczba wszystkich wygenerowanych punktów w zbiorze, a k to moc zbioru punktów należących do otoczki wypukłej.

5.1.1. Algorytm Jarvisa

Algorytm Jarvisa (czasami nazywany metodą owijania prezentu) pozwala na wyznaczanie otoczki wypukłej punktów umieszczonych na płaszczyźnie lub przestrzeni o większej liczbie wymiarów (w szczególności pozwala na znajdowanie otoczki wypukłej w przestrzeni trójwymiarowej $\mathbb{R}^3$).

Algorytm ten polega na wyznaczeniu najpierw punktu startowego P o najmniejszej współrzędnej y, a jeżeli jest kilka takich punktów to wybraniu tego o najmniejszej współrzędnej x (co ma złożoność $O(n)$) oraz punktu Q analogicznie jak punktu P, ale wybierając największe współrzędne.

Algorytm wyznacza prawy i lewy łańcuch otoczki poprzez wybieranie kolejnych punktów, dla których kąt między poprzednim i wybranym punktem spośród możliwych jest najmniejszy. Połączenie prawe i lewej otoczki daje wynikową otoczkę wypukłą.

5.1.2. Algorytm Grahama

Algorytm Grahama jest efektywnym algorytmem znajdowania otoczki wypukłej dla punktów na płaszczyźnie $\mathbb{R}^2$, jednocześnie nie istnieją warianty tego algorytmu dla przestrzeni o większej liczbie wymiarów.

Algorytm ten polega na wyznaczeniu najpierw punktu startowego P o najmniejszej współrzędnej y, a jeżeli jest kilka takich punktów to wybraniu tego o najmniejszej współrzędnej x (co ma złożoność $O(n)$). Następnym krokiem algorytmu jest posortowanie punktów według tego jaki kąt tworzy dla rozpatrywanego punktu R wektor (P, R) z dodatnim kierunkiem osi OX. Sortowania tego dokonuje się za pomocą wybranego algorytmu sortowania (o złożoności $O(n \log n)$) i obliczenia wartości wyznacznika (a w przypadku punktów tworzących ten sam kąt usunięcia wszystkich oprócz tego położonego najdalej od punktu P). Następnie przechodzi się na kolejne wierzchołki z posortowanej listy i sprawdza kierunek, w którym się przesunięto (również obliczając wartość

wyznacznika). Jeżeli poruszamy się w lewo to ustawia się rozpatrywany punkt R na stosie, jeżeli porusza się w prawo to usuwa się poprzedni punkt najwyżej na stosie przed dodaniem na stos punktu R. Jeżeli punkty są współliniowe to usuwany jest punkt najwyżej na stosie. Cała operacja przechodzenia po punktach ma złożoność $O(n-3)$, a inicjalizacja stosu. $O(1)$, zatem cały algorytm Grahama ma złożoność taką jak algorytm sortujący $O(n \log n)$.

5.2. Funkcje pomocnicze

Do wykonywania obliczeń przygotowano funkcje pomocnicze dla generatora punktów i algorytmów znajdowania otoczki wypukłej:

- det – obliczanie wyznacznika,
- orient – klasyfikacja punktu,
- quick_sort i partition – sortowanie w algorytmie Grahama,
- min_left – znajdowanie skrajnie lewego punktu zgodnie z przyjętym porządkiem,
- distance – obliczanie odległości (metryka Euklidesowa),
- timer – liczenie czasu potrzebnego do wywołania funkcji,
- *jarvis_time* – zmodyfikowana funkcja algorytmu Jarvisa, która nie generuje wizualizacji otoczki wypukłej, wykorzystana przy testach wydajnościowych,
- *graham_time* – zmodyfikowana funkcja algorytmu Grahama, która nie generuje wizualizacji otoczki wypukłej, wykorzystana przy testach wydajnościowych.

5.3. Metoda obliczenia wyznaczników

Do obliczania wyznacznika własną implementacją przygotowano funkcje wykonujące stosowne obliczenia odpowiednimi sposobami, w laboratorium wykorzystano wyznacznik 3x3.

6. Klasyfikacja punktów

6.1. Kryterium klasyfikacyjne

Punkty w zbiorach sklasyfikowano na te znajdujące się po lewej, po prawej oraz współliniowe (znajdujące się na jednej prostej) na podstawie wartości obliczonego wyznacznika.

Klasyfikacji dokonano na podstawie wartości wyznacznika dla punktów a, b i badanego punktu c. Wykorzystano następującą zależność:

Punkt c jest wobec punktów (prostej łączącej punkty) a i b:

$$\det(a,b,c)\begin{cases}(-\infty,0) \Rightarrow po\ prawej,\\ 0 \Rightarrow współliniowy,\\ (0,+\infty) \Rightarrow po\ lewej.\end{cases}$$

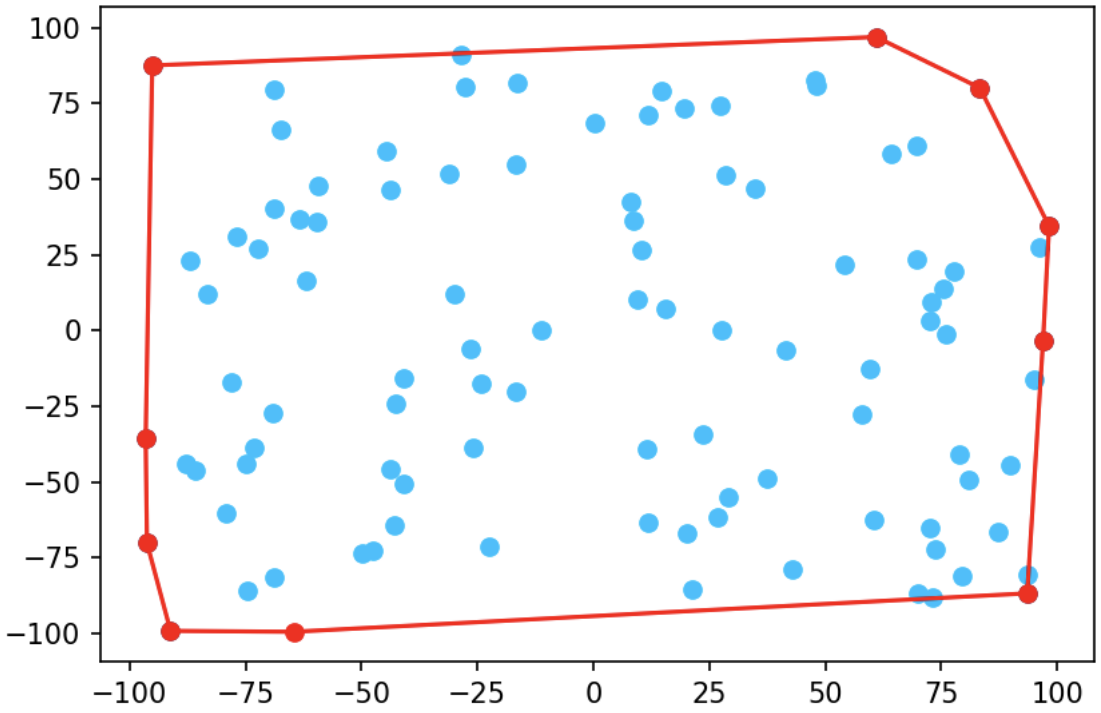
6.2. Tolerancje dla zera

Klasyfikacji dokonano z różnymi tolerancjami dla zera (oznaczanymi jako *epsilon*). Wykorzystanie tolerancji związane jest z reprezentacją liczb rzeczywistych w komputerze i wynikającym z tego obarczeniem wyników obliczeń niedokładnością przy wykorzystaniu liczb zmiennoprzecinkowych (w języku *Python* typ zmiennej *float* – 64-bitowa liczba zmiennoprzecinkowa).

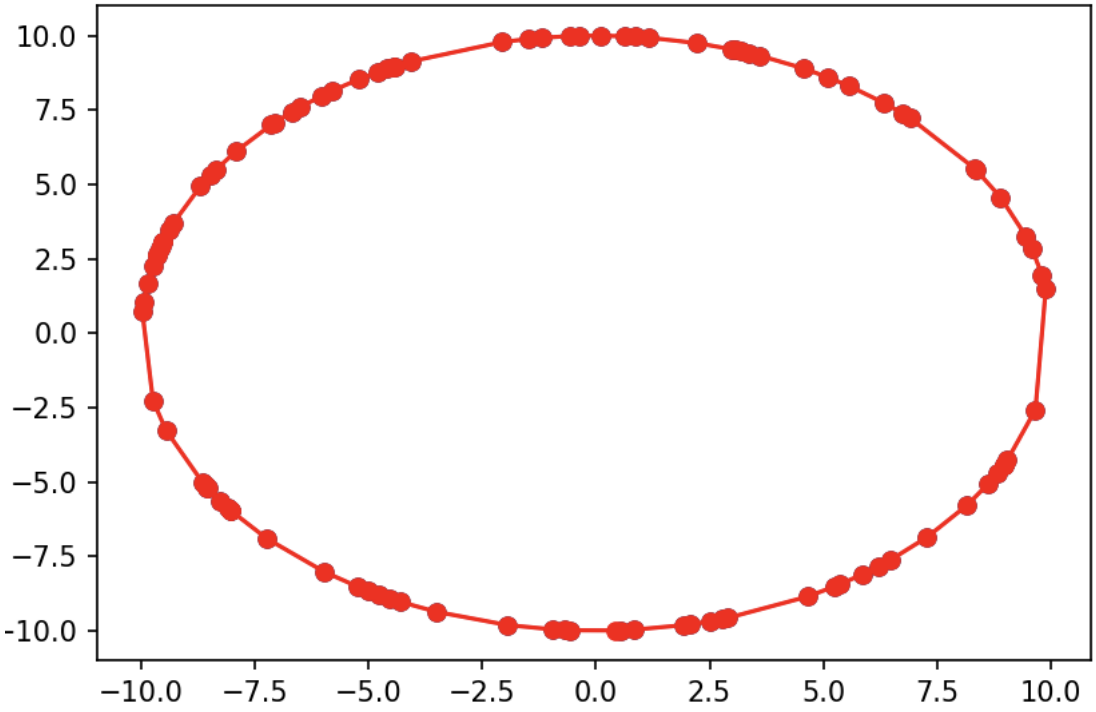
Dokonano obliczeń dla tolerancji:

$$\varepsilon = 10^{-12}.$$

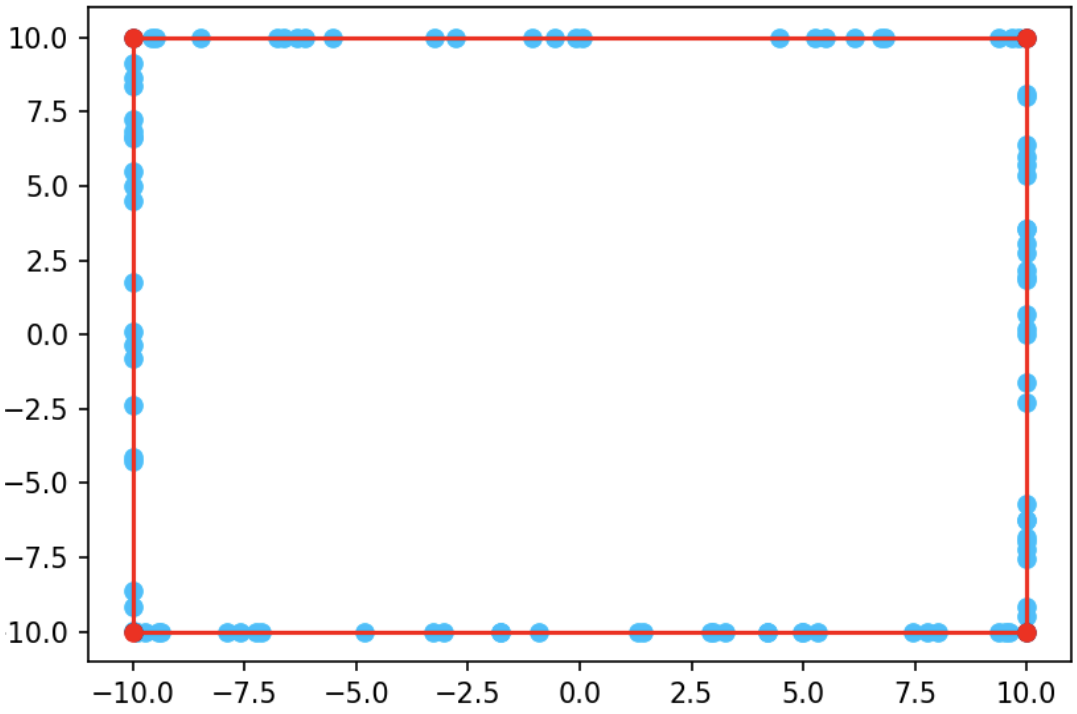
7. Analiza wyników



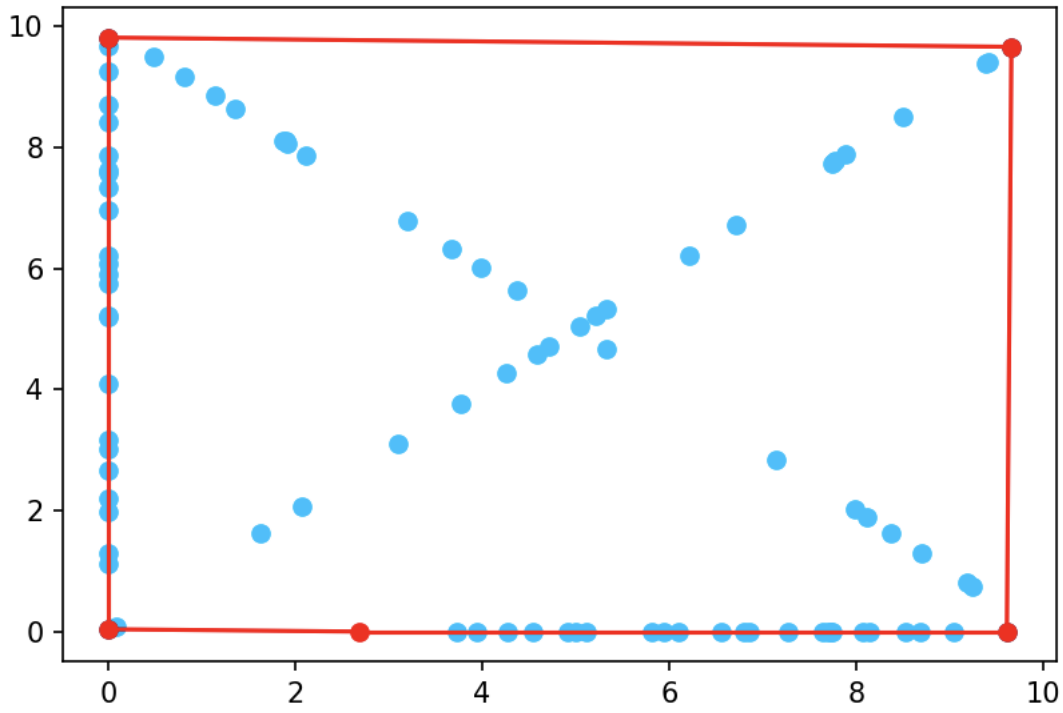
Rysunek 7 a) Zbiór A



Rysunek 7 b) Zbiór B



Rysunek 7 c) Zbiór C



Rysunek 7 d) Zbiór D

Wyniki były takie same dla obu algorytmów.

8. Testy wydajnościowe

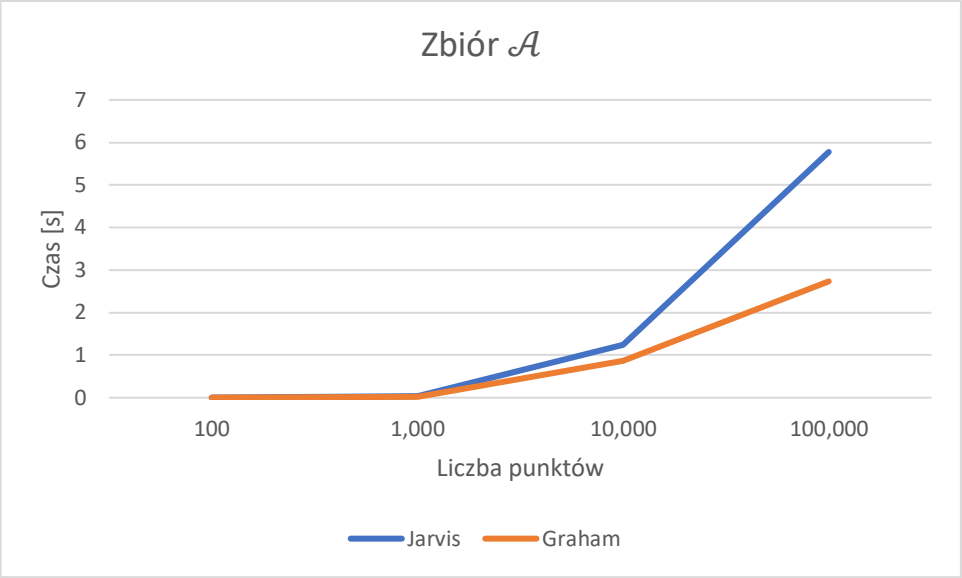
W laboratorium wykonano także testy wydajnościowe polegające na sprawdzeniu czasu potrzebnego do wykonania funkcji znajdujących otoczkę wypukłą (do tego zadania wykorzystano wcześniej omówione funkcje *jarvis_time* i *graham_time*).

Do badania wykorzystano wszystkie punkty z różną liczbą punktów oraz przyjęto $\varepsilon = 10^{-12}$, zbadano oba algorytmy (Jarvisa i Grahama).

Wyniki przybliżono do czwartego miejsca po przecinku i zaprezentowano w tabeli:

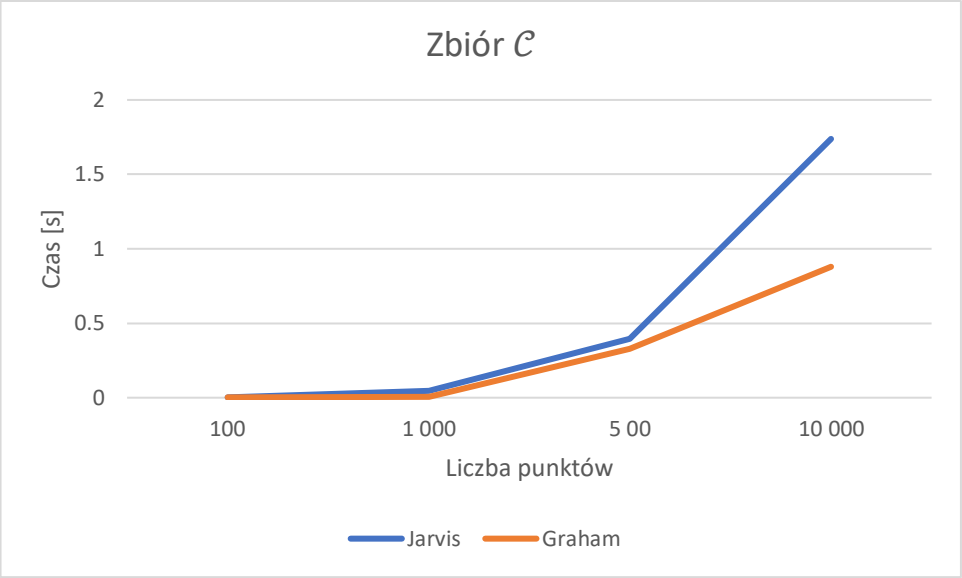
Lp.	Obiekt testowy		Czas potrzebny na wykonanie algorytmu [s]	
	Zbiór	Liczba punktów	Jarvis	Graham
1.	$\mathcal{A}$	100	0,0026	0,0017
2.	$\mathcal{A}$	1 000	0,0371	0,0186
3.	$\mathcal{A}$	10 000	1,2470	0,8742
4.	$\mathcal{A}$	100 000	5,7758	2,7344
5.	$\mathcal{B}$	100	0,0162	0,0036
6.	$\mathcal{B}$	1 000	1,3673	0,0181
7.	$\mathcal{B}$	2 000	7,5720	0,3172
8.	$\mathcal{B}$	5 000	36,6219	0,9667
9.	$\mathcal{C}$	100	0,0022	0,0021
10.	$\mathcal{C}$	1 000	0,0475	0,0068
11.	$\mathcal{C}$	5 000	0,3967	0,1440
12.	$\mathcal{C}$	10 000	1,738	0,8793
13.	$\mathcal{D}$	(25, 20)	0,0014	0,0028
14.	$\mathcal{D}$	(50, 40)	0,0019	0,0035
15.	$\mathcal{D}$	(25 000, 20 000)	0,6846	4,1842
16.	$\mathcal{D}$	(50 000, 40 000)	1,3495	9,2173

Tabela 8

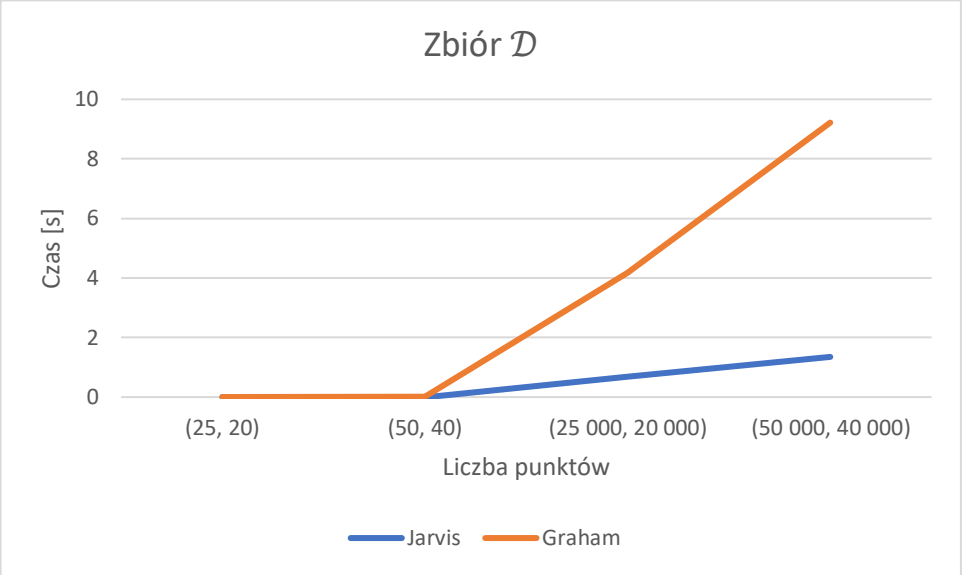


Wykres 8 a) Prezentacja wydajności algorytmów dla zbioru $\mathcal{A}$

Wykres 8 b) Prezentacja wydajności algorytmów dla zbioru $\mathcal{B}$



Wykres 8 c) Prezentacja wydajności algorytmów dla zbioru $\mathcal{C}$



Wykres 8 d) Prezentacja wydajności algorytmów dla zbioru $\mathcal{D}$

9. Podsumowanie i wnioski

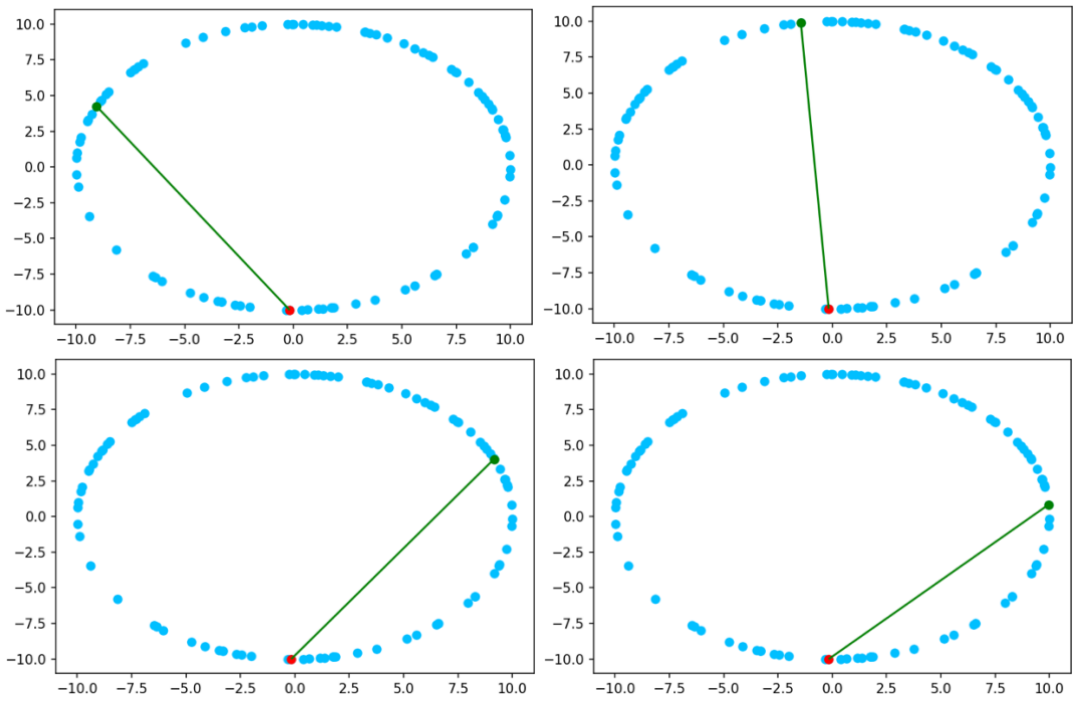
9.1. Podsumowanie

Wyniki laboratorium ukazują jak ważna jest złożoność algorytmów użyta w projektach informatycznych oraz dopasowanie wykorzystanych algorytmów do danego projektu. Największe różnice między algorytmami widoczne są w zbiorze $\mathcal{B}$ oraz $\mathcal{D}$.

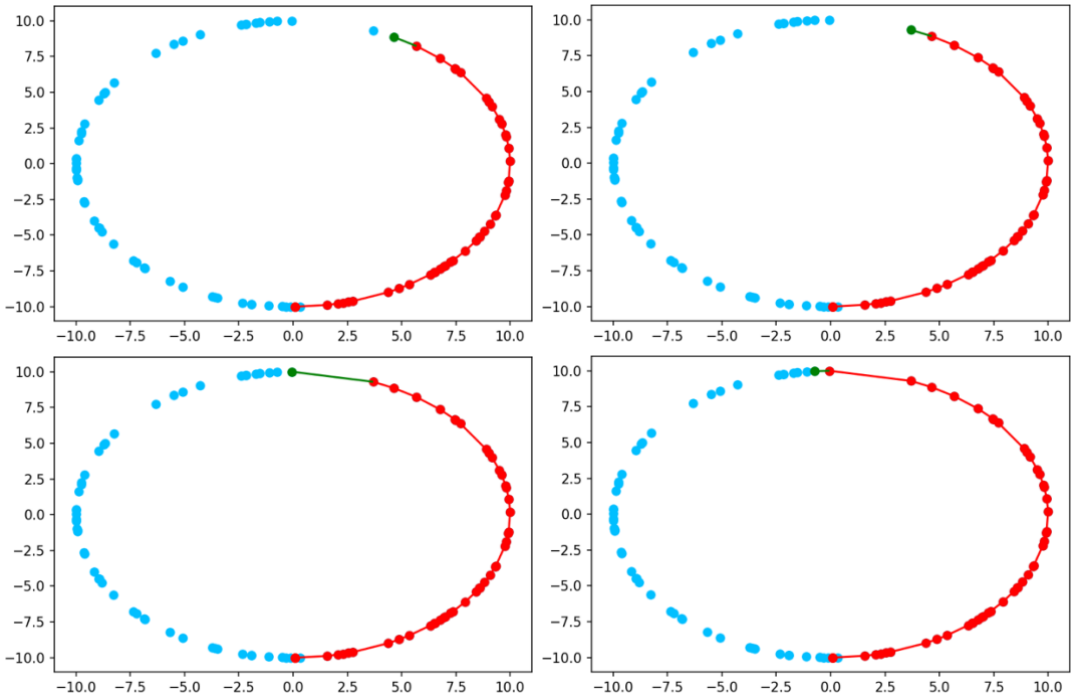
Algorytm Grahama okazał się ogólnie szybszy od algorytmu Jarvisa w 3 przypadkach na 4 Algorytm Grahama ma złożoność $O(n\ log\ n)$, a złożoność algorytmu Jarvisa to $O(kn)$, gdzie k to liczba wierzchołków otoczki wypukłej, k w większości przypadków jest dużo mniejsze od n, natomiast w pesymistycznym przypadku – gdy k bliskie jest n – złożonośc tego algorytmu to $O(n^2)$.

9.2. Przypadek zbioru $\mathcal{B}$

W zbiorze $\mathcal{B}$ (okrąg) algorytm Grahama po sortowaniu przechodził po kolejnych punktach otoczki, natomiast algorytm Jarvisa wykonywał dla każdego punktu dodatkowe wyszukiwania kolejnego punktu otoczki. Algorytm Grahama był w tym przypadku zdecydowanie szybszy od algorytmu Jarvisa. W algorytmie Grahama największą złożoność – która determinuje złożoność całkowitą algorytmu – to sortowanie. Wykonujemy je algorytmem sortowania szybkiego (*quick sort*), którego złożoność możemy określić na O($n\ log\ n$).



Rysunek 9.2. a) Prezentacja kilku kolejnych kroków działania algorytmu Jarvisa w zbiorze

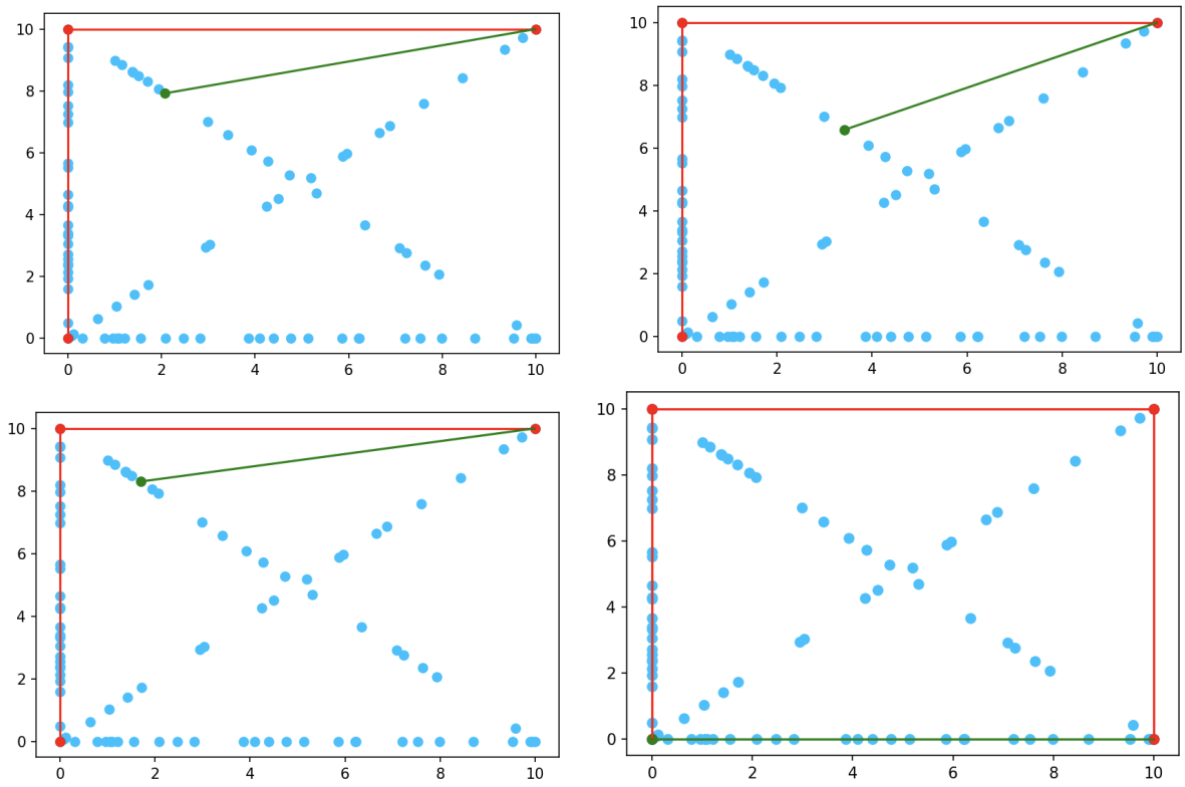


Rysunek 9.2. b) Prezentacja kilku kolejnych kroków działania algorytmu Grahama w zbiorze $\mathcal{B}$

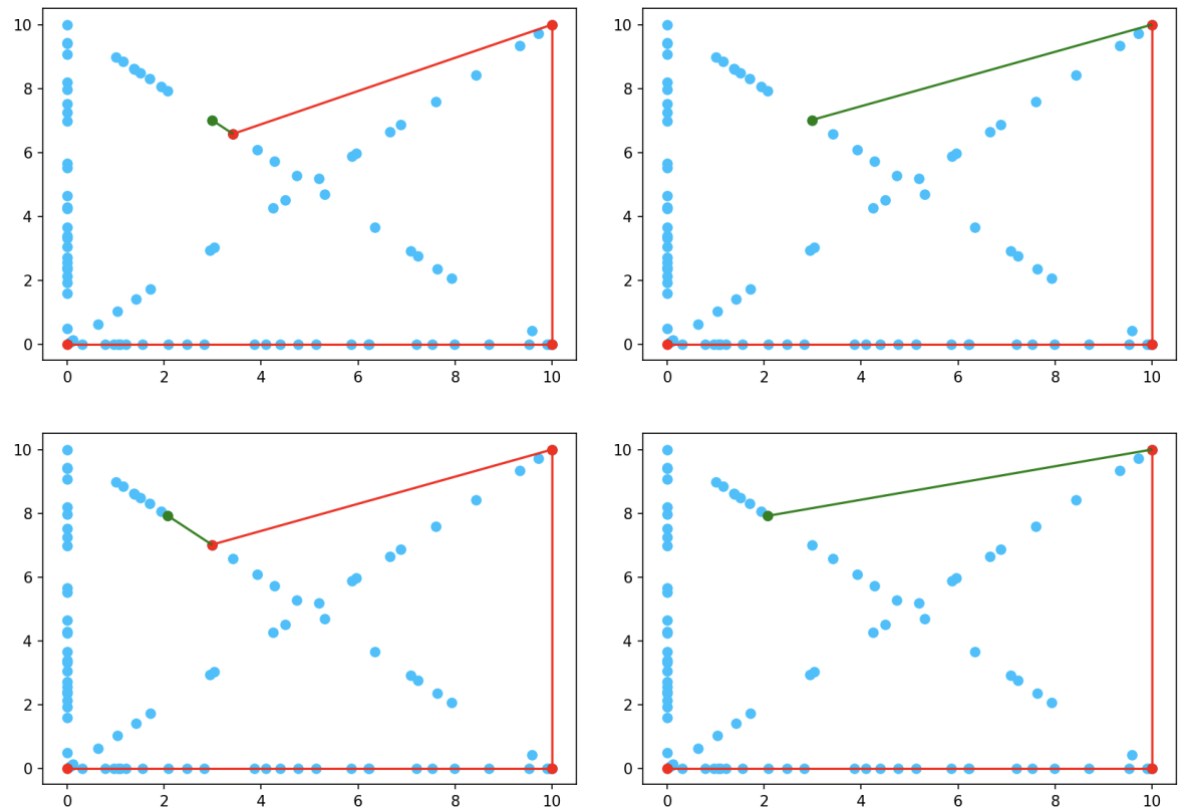
9.3. Przypadek zbioru $\mathcal{D}$

W zbiorze $\mathcal{D}$ szybszy okazał się algorytm Jarvis pomimo swojej teoretycznej większej złożoności obliczeniowej. Algorytm Jarvisa był tym przypadku szybszy ze względu na specyfikacje zbioru $\mathcal{D}$. W tym zbiorze algorytm Grahama kwalifikował tymczasowo punkty na przekątnych jako potencjalne punkty na otoczkach i dalej szukał kolejnych punktów potencjalnie należących do toczki, natomiast algorytm Jarvisa sprawdzał kolejne punkty i odrzucał punkty na przekątnych, które na otoczce znaleźć się nie mogły.

Tymczasowe kwalifikowanie punktów na przekątnych jako potencjalnie znajdujących się na otoczce w algorytmie Grahama sprawiło, że dużo szybszy okazał się algorytm Jarvisa. Ukazuje to, że algorytmy powinno się dobierać indywidualnie do danej sytuacji, nie kierując się jedynie teoretyczną złożonością „na papierze”.



Rysunek 9.3. a) Prezentacja kilku kolejnych kroków działania algorytmu Jarvisa w zbiorze $\mathcal{D}$



Rysunek 9.3. b) Prezentacja kilku kolejnych kroków działania algorytmu Grahama w zbiorze $\mathcal{D}$

9.4. Wnioski

Wnioski te są szczególnie cenne dla projektów informatycznych, w których precyzja obliczeń i identyfikacja pewnych danych z wykorzystaniem algorytmów geometrycznych są sprawą kluczową.

* * *