Algorytmy geometryczne - Ćwiczenie 2. Paulina Jędrychowska

1. Dane techniczne urządzenia i wykorzystane narzędzia:

System Windows 10 x64

Procesor i5-9300H

Pamięć RAM 16GB

Środowisko Jupyter Notebook

Język programowania Python 3.0

2. Generowanie punktów:

Wszystkie obliczenia były robione na współrzędnych rzeczywistych podwójnej precyzji.

Do wygenerowania były cztery zbiory punktów. Wygenerowane punkty zostały umieszczone w osobnych tablicach:

a) Zbiór punktów 1 - 100 losowo wygenerowanych punktów o współrzędnych z przedziału [-100, 100],

b) Zbiór punktów 2 - 100 losowo wygenerowanych punktów leżących na okręgu o środku (0,0) i promieniu R=10,

c) Zbiór punktów 3 - 100 losowo wygenerowanych punktów leżących na bokach prostokąta

o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10),

d) Zbiór punktów 4 - 94 punkty: wierzchołki kwadratu (0, 0), (10, 0), (10, 10), (0, 10); po 25 punktów na dwóch bokach kwadratu leżących na osiach; po 20 punktów na przekątnych kwadratu

Do wygenerowania zbiorów punktów została użyta funkcja random.uniform(a, b), która generuje liczby typu double z zakresu [a, b].

Dla okręgu b) zostały również użyte funkcje pi, sin i cos z biblioteki math. Punkty na okręgu zostały wygenerowane poprzez stworzenie punktów reprezentujących kąt alfa z zakresu [0, 2π] i podstawienie odpowiednio: x = 10*cos(alfa) i y = 10*sin(alfa).

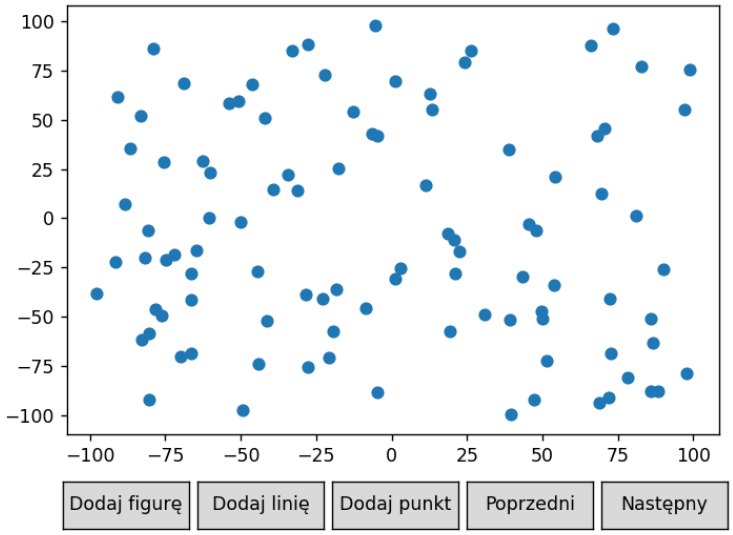
Dla prostokąta c) za pomocą funkcji random.randint() (która generuje liczby całkowitoliczbowe z danego zakresu) losowano bok na którym ma się znaleźć punkt. Boki

prostokąta zostały wyznaczone jako proste: x = -10, x = 10, y = -10, y = 10. Następnie dla wylosowanego boku losowano drugą współrzędną z przedziału [-10, 10].

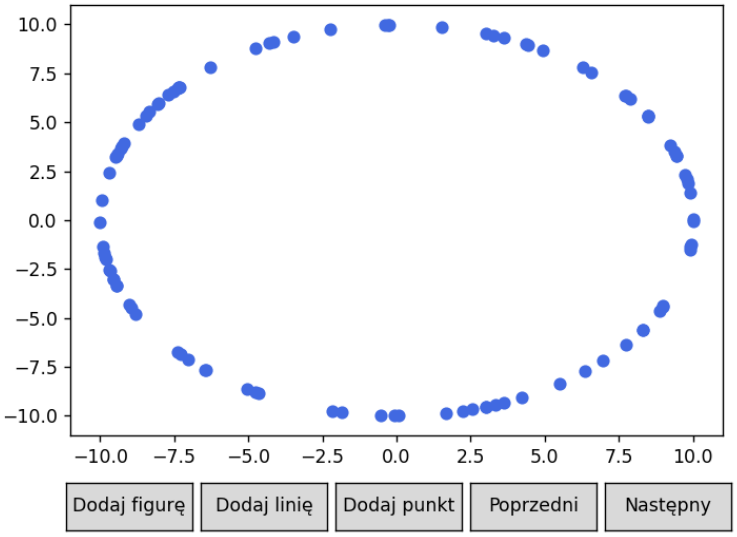
Dla kwadratu d) punkty (0, 0), (10, 0), (10, 10), (0, 10) zostały dodane na sztywno. Boki kwadratu zostały wyznaczone ze wzorów: x = 0, y = 0. Druga współrzędna losowana była ze zbioru [0, 10]. Do wyznaczenia punktów na przekątnych, dla wylosowanej współrzędnej x ze zbioru [0, 10] współrzędna y została wyznaczona ze wzorów y = x, y = -x.

Wszystkie zbiory punktów zostały zwizualizowane przez dostarczone narzędzie graficzne oparte o bibliotekę matplotlib.

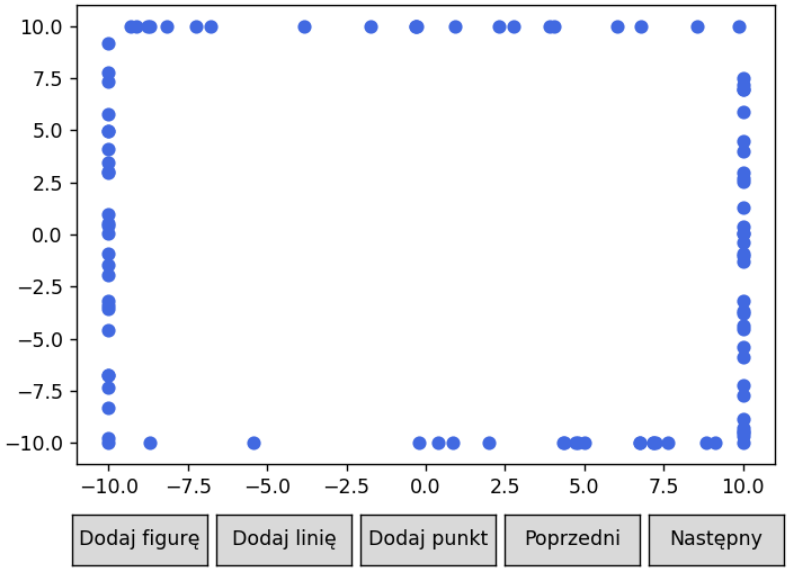
Wykres 1.1. Zbiór punktów 1



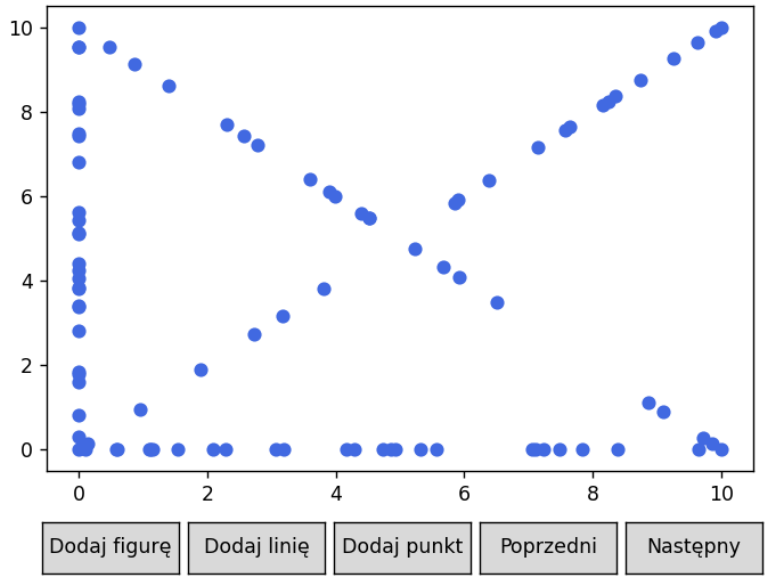
Wykres 1.2. Zbiór punktów 2



Wykres 1.3. Zbiór punktów 3



Wykres 1.4. Zbiór punktów 4



Program umożliwia również generowanie zbiorów punktów o zadanych parametrach. Są one generowane w ten sam sposób co powyższe zbiory, ale zamiast sztywno określonych parametrów przyjmują zmienne.

3. Obliczanie wyznacznika:

Do wyliczenia wyznacznika użyto wyznacznika 3x3 własnej implementacji.

Wzór 1. Wyznacznik 3x3

$$\det(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix}$$

Funkcja orient() zwraca znak wyznacznika, przy określonej tolerancji.

4. Tolerancja dla zera:

Jako tolerancję dla zera przyjęto 10^{-12}, gdyż dla 10^{-14} algorytm błędnie nie zaliczał niektórych punktów do prostej. Przez co błędnie wyznaczał otoczkę, zaliczając do niej punkty współliniowe.

5. Działanie algorytmu Grahama:

Algorytm działa na kopii tablicy punktów, aby usuwając wierzchołki nie modyfikować wejściowego zbioru punktów. Przez co algorytm zajmuje dodatkowe O(n) pamięci, ale nie stanowi to problemu, gdyż stos w pesymistycznym przypadku też zajmuje O(n) pamięci.

Na początku wybierany jest wierzchołek o najmniejszej współrzędnej y (jeśli jest kilka o tej samej współrzędnej to brana jest pod uwagę współrzędna x). Następnie wierzchołki są sortowane za pomocą funkcji quicksort(). Punkty są sortowane względem kąta jaki tworzy wektor najmniejszego punktu oraz rozważanego punktu względem osi x. Jeśli punkty są współliniowe, to jako mniejszy uznajemy punkt bardziej oddalony od punktu najmniejszego, a drugi punkt (mniej oddalony) dodajemy do tablicy removed, która później zostanie usunięta. Porównywanie ze sobą dwóch punktów przebiega w funkcji compare(), która sprawdza znak ich wyznacznika z punktem startowym.

Po posortowaniu usuwane są wszystkie wierzchołki, które zostały uznane za współliniowe. Następnie na stos dodane są 3 początkowe punkty.

Główna pętla algorytmu przebiega po wszystkich wierzchołkach. W pętli sprawdzana jest położenie danego wierzchołka względem prostej (złożonej z dwóch ostatnich punktów na stosie). Do porównania punktów ponownie sprawdzano znak wyznacznika. Jeżeli punkt znajdował się na lewo od prostej był on dodawany na stos. Dla punktów leżących na prawo ściągano ostatni element ze stosu. Dla punktów współliniowych ze stosu usuwano ostatni punkt i na jego miejsce dawano rozpatrywany (bardziej oddalony) punkt.

Po zakończeniu pętli sprawdzane jest czy ostatni punkt nie jest współliniowy z punktem pierwszym i przedostatnim.

5. Działanie algorytmu Jarvisa:

Na początek tak jak w algorytmie Grahama wybieramy najmniejszy wierzchołek. Do listy wierzchołków otoczki dodajemy sztuczny punkt, dla którego prosta pomiędzy nim a wierzchołkiem startowym jest równoległa do osi x. Posłuży ona za wstępną pierwszą krawędź otoczki względem, której będą kategoryzowane kolejne punkty.

Główna pętla algorytmu przebiega, dopóki rozważanym wierzchołkiem nie będzie z powrotem wierzchołek startowy. W pętli sprawdzamy wszystkie wierzchołki i wybieramy ten z nich który tworzy najmniejszy kąt (liczony zgodnie z ruchem wskazówek zegara) z prostą złożoną z dwóch ostatnio dodanych wierzchołków otoczki.

Na koniec usuwany jest sztuczny wierzchołek dodany na początek listy wynikowej.

6. Przedstawienie przebiegu algorytmów:

Legenda oznaczeń wykresów :

- Granatowy – punkty nie rozważane w danej iteracji, nie należące do otoczki
- Czerwony – punkty usunięte po sortowaniu w algorytmie Grahama
- Zielony – punkty należące do otoczki
- Cyjanowy – punkt aktualnie rozważany

Krawędzie otoczki oraz krawędzie aktualnie rozpatrywane zostały zaznaczone odpowiadającymi im kolorami.

Oba algorytmy dla przyjętej tolerancji poprawnie wyznaczały otoczkę. Dla obu algorytmów wyznaczone otoczki składały się dokładnie z tych samych punktów.

a) Zbiór punktów 1:

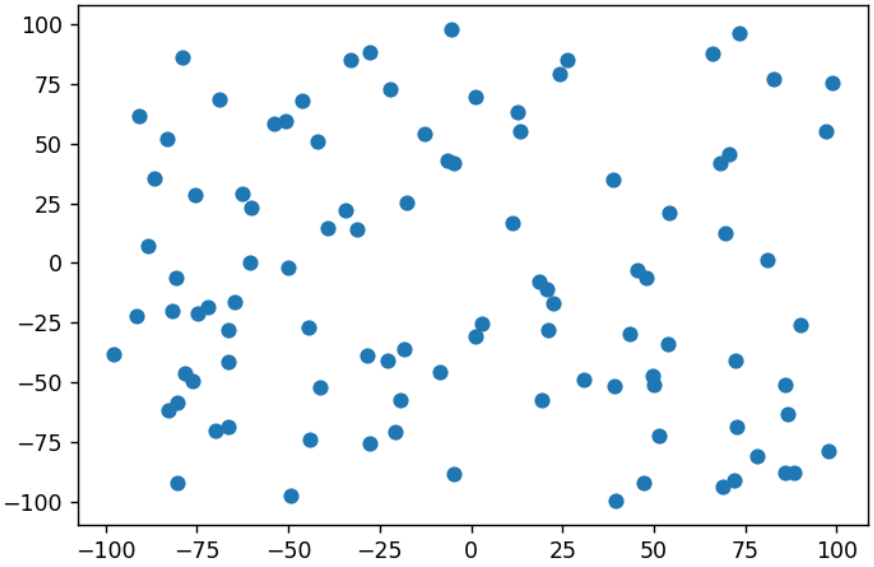
Dla tego zbioru punktów algorytmy nie miały większego problemu z wyznaczeniem otoczki. Rozmiar otoczki był dość mały względem ilości punktów. Dzięki temu, że w tym zbiorze znajdują się losowo generowane punkty w nieokreślonych miejscach możemy zauważyć uniwersalność znajdowania otoczki.

- Algorytm Grahama

Dla zbioru 1 w algorytmie Grahama rzadko kiedy przy sortowaniu znajdowano punkty współliniowe. Jak widać dla tego zbioru nie wykryto żadnych punków współliniowych.

Wykres 2.1. Posortowane punkty w algorytmie Grahama

dla zbioru punktów 1

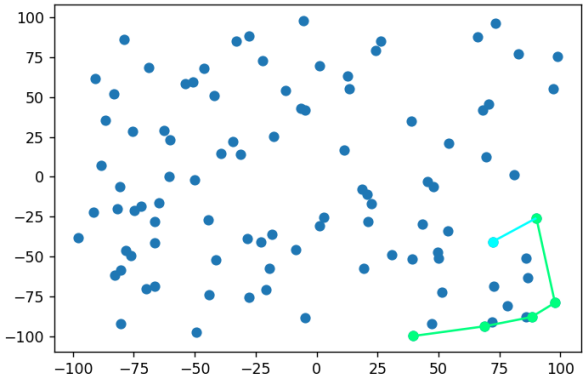


Do pokazania głównej części algorytmu została wybrana iteracja, dla której najwyraźniej widać przebieg.

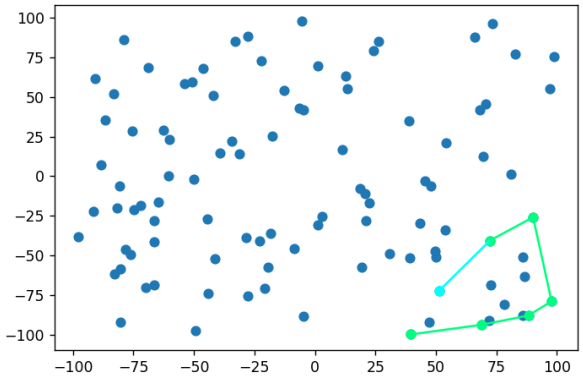
Wykresy 2.2. Główna część algorytmu Grahama

dla zbioru 1

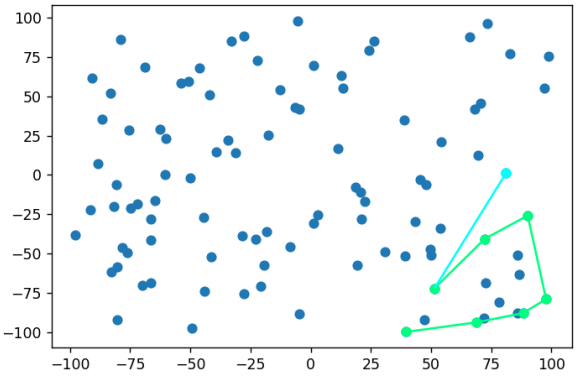
Krok 1



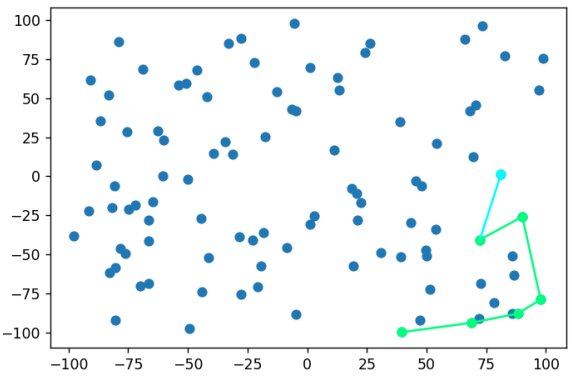
Krok 2



Krok 3

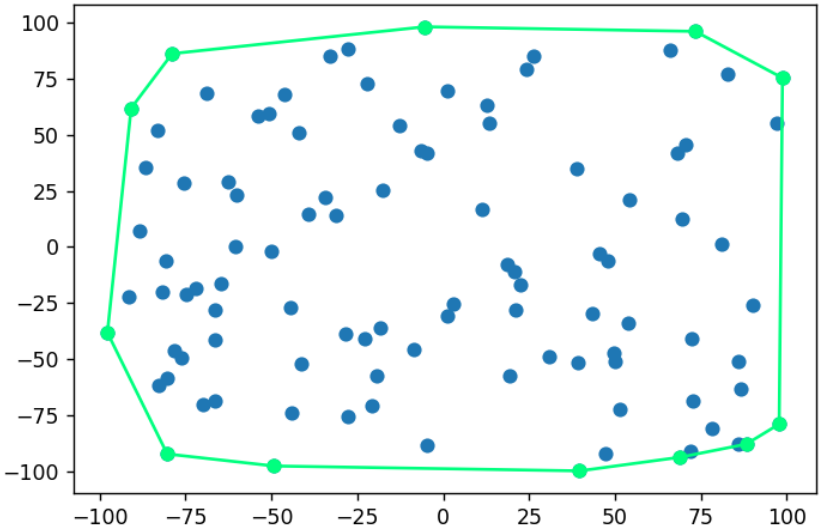


Krok 4



Wykres 2.3. Otoczka wypukła wyznaczona przez algorytm Grahama

dla zbioru 1

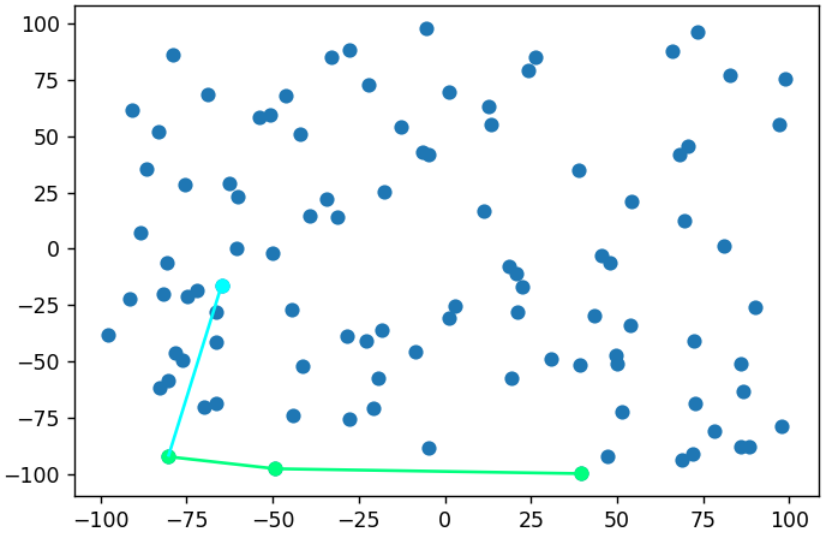


- Algorytm Jarvisa

W głównej części algorytmu rozważane są wszystkie wierzchołki i z nich wybierany jest ten pod najmniejszym kątem do ostatniego boku otoczki.

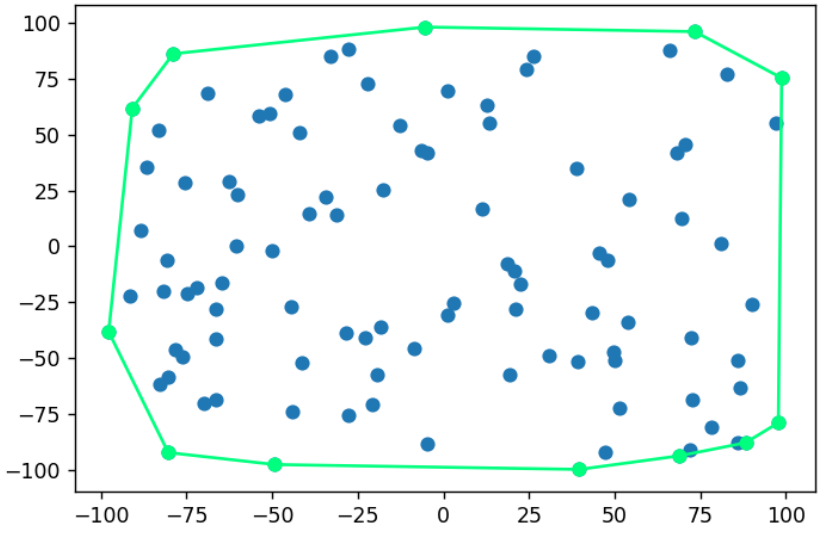
Wykres 2.4. Główna część działania algorytmu Jarvisa

dla zbioru punktów 1



Wykres 2.5. Otoczka wypukła wyznaczona przez algorytm Jarvisa

dla zbioru 1



b) Zbiór punktów 2:

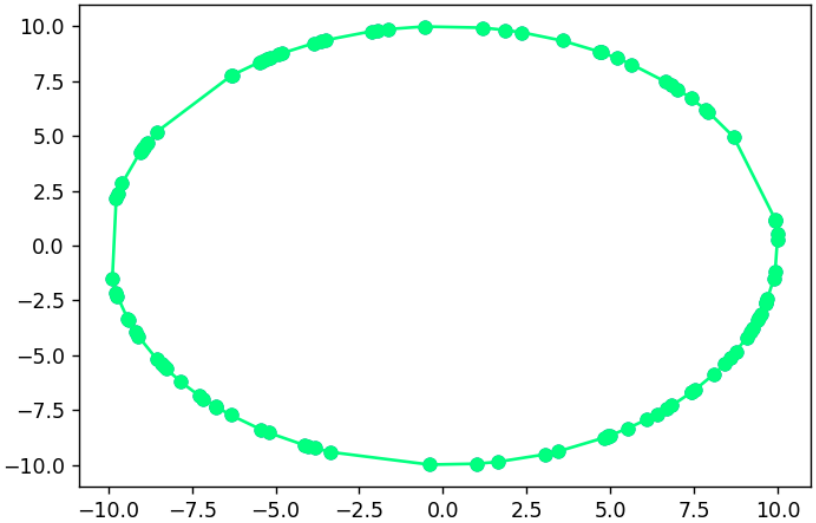
Dla tego zbioru punktów do otoczki należały wszystkie punkty ze zbioru. Oba algorytmy poprawnie wyznaczyły otoczkę, ale algorytm Jarvisa robił do znacznie dłużej. Zaproponowano implementację tego zbioru prawdopodobnie właśnie, aby zauważyć znacznie dłuższy czas działania algorytmu Jarvisa dla zbiorów z dużą ilością punktów należących do otoczki.

- Algorytm Grahama

Po sortowaniu żadne punkty nie są usuwane, co się zgadza ponieważ na okręgu żadne punkty nie są współliniowe. Następnie dzięki posortowaniu rozpatrywane są kolejne wierzchołki i każdy z nich jest dodawany do otoczki.

Wykres 3.1. Otoczka wypukła wyznaczona przez algorytm Grahama

dla zbioru 2

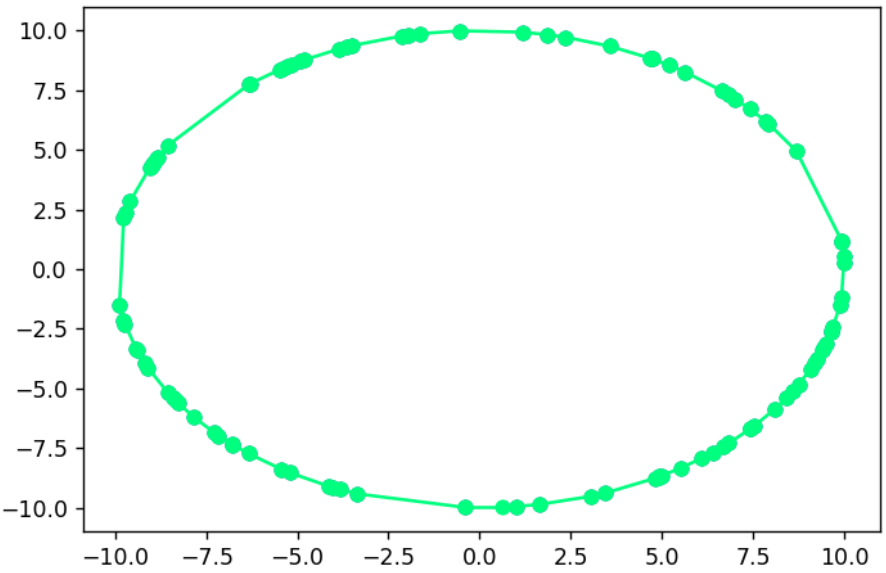


- Algorytm Jarvisa

Algorytm Jarvisa, po znalezieniu punktu należącego do otoczki rozpatruje wszystkie punkty należące do zbioru, z tego powodu im więcej punktów w otoczce tym dłuższy czas działania algorytmu. Nie ma znaczenia kształt otoczki, a jedynie ilość wierzchołków do niej należących.

Wykres 3.2. Otoczka wypukła wyznaczona przez algorytm Jarvisa

dla zbioru 2



c) Zbiór punktów 3:

Dla tego zbioru punktów do otoczki należało zazwyczaj 8 punktów, aczkolwiek teoretycznie ich liczba może wynosić od 4 do 8. Zazwyczaj zaliczane do otoczki są po dwa skrajne wierzchołki z każdego boku. Natomiast przy wygenerowaniu punktów blisko wierzchołków kwadratu algorytm mógł je zaliczać jako skrajne punkty, wtedy liczba zaliczonych punktów będzie mniejsza. Zestaw ten został zaproponowany, aby przetestować radzenie sobie algorytmu z punktami współliniowymi.

- Algorytm Grahama

Algorytm usuwa współliniowe wierzchołki znalezione na etapie sortowania. Przez położenie skrajnych punktów usuwane są wszystkie wierzchołki leżące na dolnej krawędzi (oprócz punktów skrajnych), a jeżeli wylosowany punkt startowy jest lewym

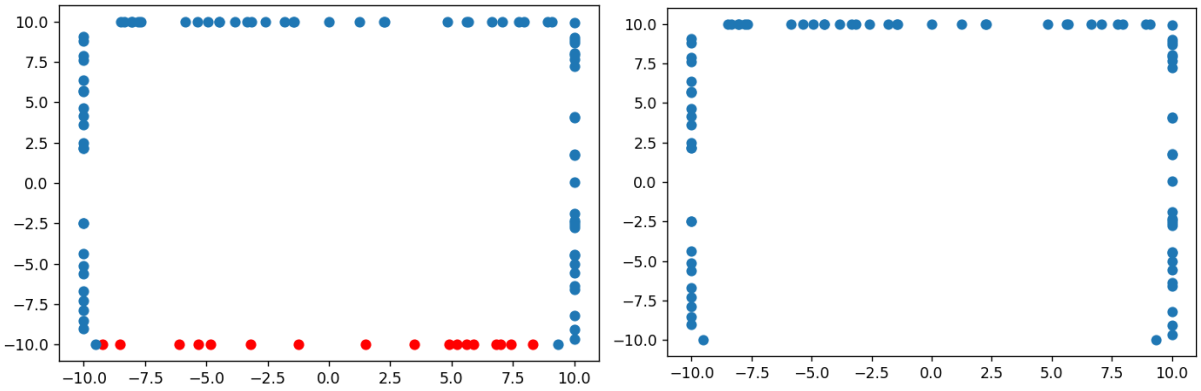
dolnym wierzchołkiem kwadratu to dodatkowo usuwane są wierzchołki z lewej krawędzi kwadratu.

Na poniższym wykresie można zaobserwować, że zostały usunięte wierzchołki z dolnego boku kwadratu, na którym znajdował się wierzchołek minimalny.

Wykres 4.1. Posortowane punkty w algorytmie Grahama

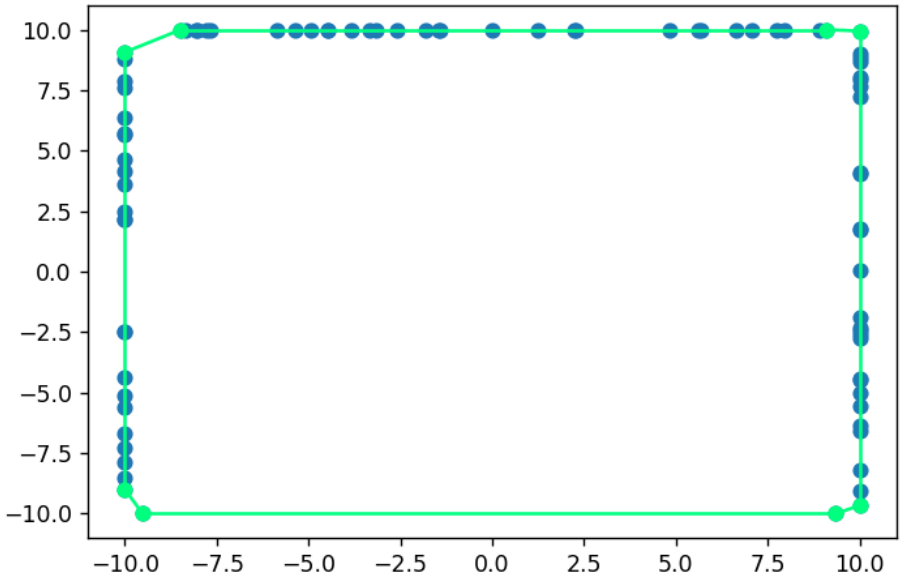
oraz zbiór po usunięciu punktów współliniowych

dla zbioru 3



Wykres 4.2. Otoczka wypukła wyznaczona przez algorytm Grahama

dla zbioru 3

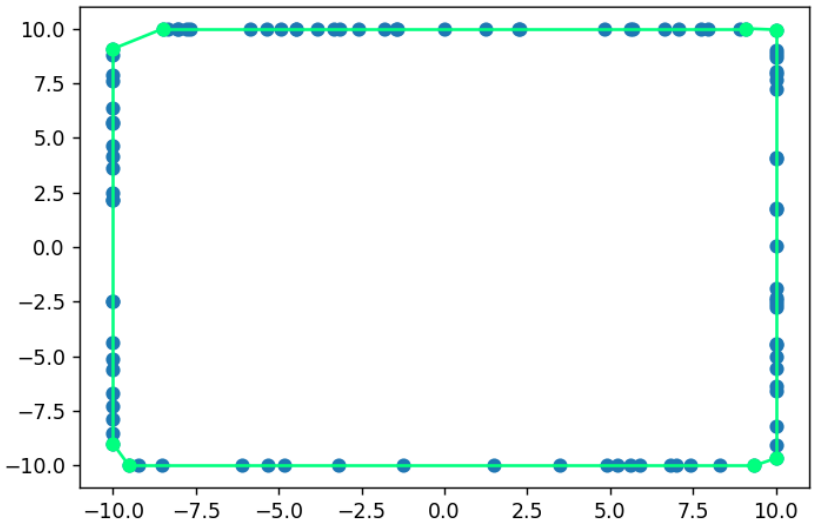


- Algorytm Jarvisa

Algorytm Jarvisa dla tego zbioru punktów działa bardzo szybko, bo będzie wykonywał tylko 8 iteracji pętli. Algorytm dobrze sobie radzi z wykrywaniem współliniowych wierzchołków i wybieraniem tego o bardziej oddalonego.

Wykres 4.3. Otoczka wypukła wyznaczona przez algorytm Jarvisa

dla zbioru 3



c) Zbiór punktów 4:

Dla tego zbioru punktów do otoczki zawsze należały 4 punkty, gdyż skrajne punkty były ustawione na sztywno. Zestaw ten został również zaproponowany, aby przetestować radzenie sobie algorytmu z punktami współliniowymi.

- Algorytm Grahama

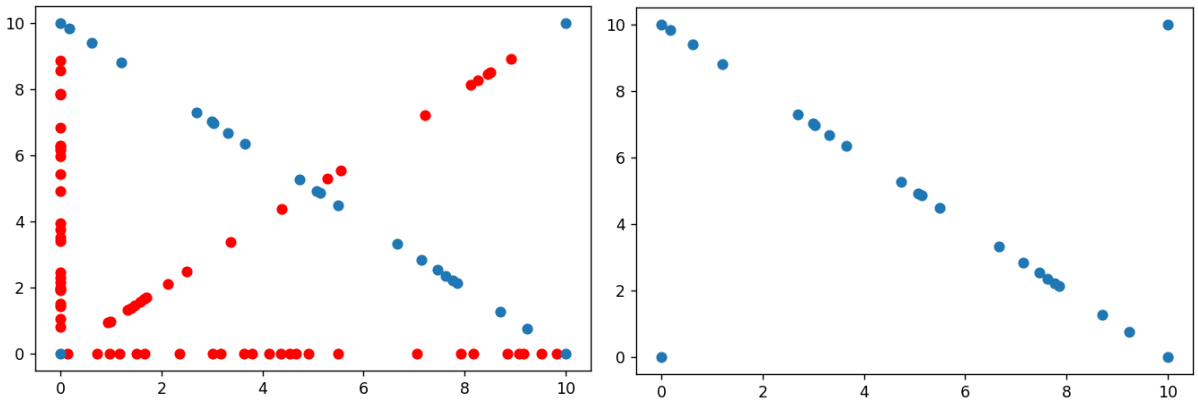
Dla tego zbioru punktów można dobrze zauważyć poszczególne etapy działania algorytmu Grahama.

Przy sortowaniu usuwane były wszystkie punkty (oprócz punktów skrajnych) z obu krawędzi kwadratu i jednej z przekątnych.

Wykres 5.1. Posortowane punkty w algorytmie Grahama

oraz zbiór po usunięciu punktów współliniowych

dla zbioru 4

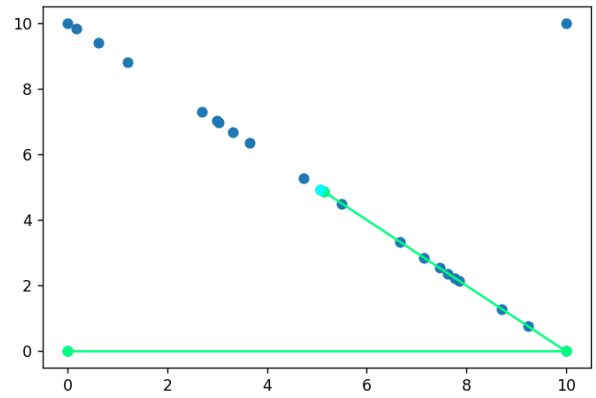


Na danym zbiorze, dobrze widoczny jest kluczowy krok algorytmu Grahama, związany ze skrętem w prawo.

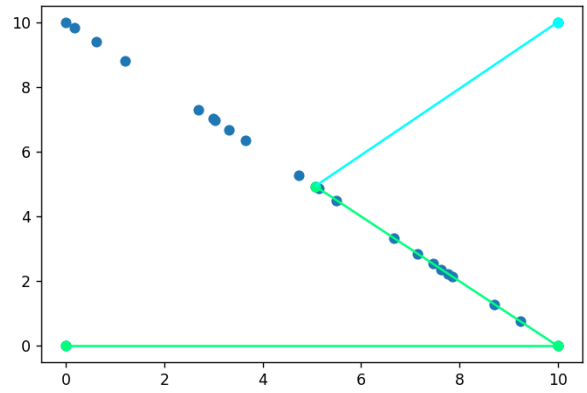
Wykresy 5.2. Główna część algorytmu Grahama

dla zbioru 4

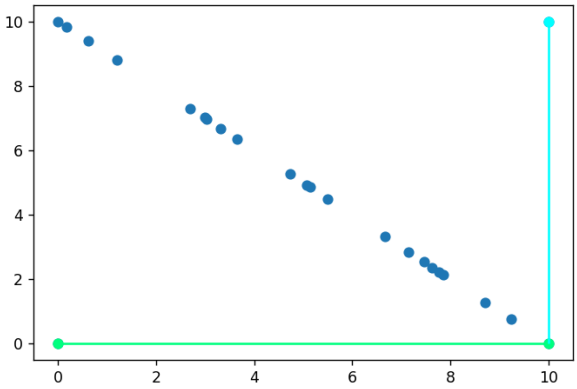
Krok 1



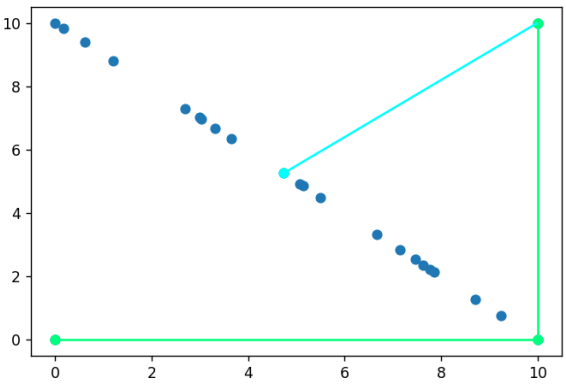
Krok 2



Krok 3

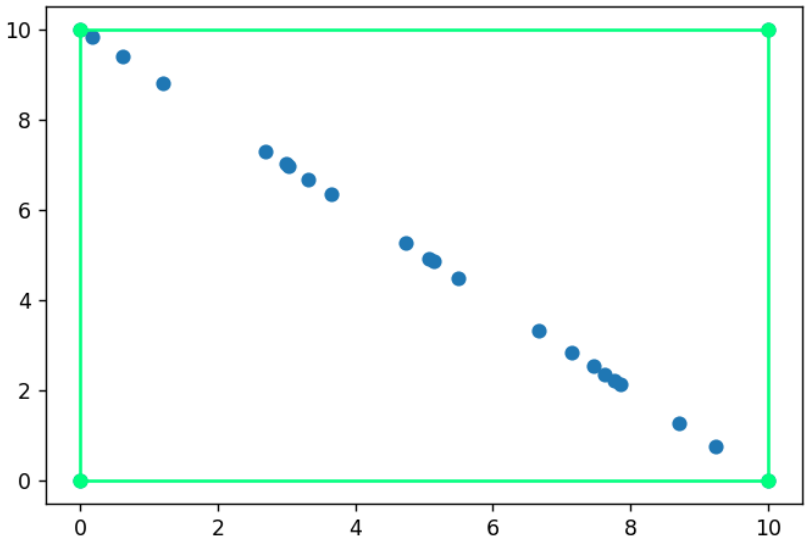


Krok 3



Wykres 5.3. Otoczka wypukła wyznaczona przez algorytm Grahama

dla zbioru 4

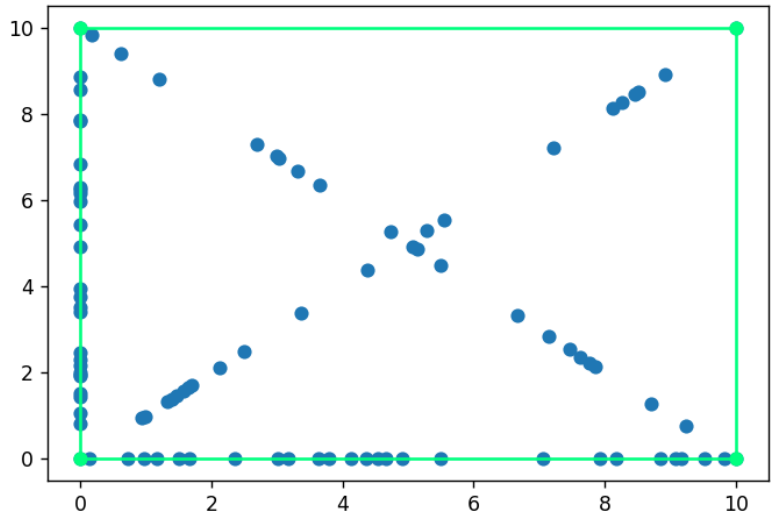


- Algorytm Jarvisa

Tak jak dla poprzedniego zbioru punktów algorytm Jarvisa działa bardzo szybko, bo będzie wykonywał tylko 4 iteracji pętli. Sytuacja jest właściwie analogiczna jak dla poprzedniego zbioru.

Wykres 5.4. Otoczka wypukła wyznaczona przez algorytm Jarvisa

dla zbioru 4



7. Porównanie czasów działania algorytmów:

Tabela 1.1. Porównanie czasów działania algorytmów Grahama i Jarvisa względem liczby punktów dla zbioru 1

Algorytm	100 punktów	1000 punktów	5000 punktów	10 000 punktów	100 000 punktów
Graham	0.0010 s	0.0170 s	0.1127 s	0.2593 s	3.1625 s
Jarvis	0.0010 s	0.0309 s	0.1875 s	0.4009 s	3.4179 s

Tabela 1.2. Porównanie czasów działania algorytmów Grahama i Jarvisa względem liczby punktów dla zbioru 2

Algorytm	100 punktów	1000 punktów	5000 punktów	10 000 punktów
Graham	0.0010 s	0.0170 s	0.1077 s	0.2304 s
Jarvis	0.0140 s	1.5578 s	39.2490 s	154.6833 s

Tabela 1.3. Porównanie czasów działania algorytmów Grahama i Jarvisa względem liczby punktów dla zbioru 3

Algorytm	100 punktów	1000 punktów	5000 punktów	10 000 punktów	100 000 punktów
Graham	0.0010 s	0.0189 s	0.1536 s	0.3231 s	9.5584 s
Jarvis	0.0020 s	0.0160 s	0.0718 s	0.1426 s	1.5598 s

Tabela 1.4. Porównanie czasów działania algorytmów Grahama i Jarvisa względem liczby punktów dla zbioru 4

Algorytm	100 punktów	1000 punktów	5000 punktów	10 000 punktów	100 000 punktów
Graham	0.0010 s	0.0309 s	0.3161 s	0.9495 s	127.8211 s
Jarvis	0,0010 s	0.0059 s	0.3900 s	0.0728 s	0.7370 s

- Złożoności czasowe algorytmów:

Graham – O(nlong)

Jarvis – O(nk)

Gdzie n to liczba wierzchołków, a k to liczba punktów należących do otoczki.

Dla pierwszego, losowego zestawu danych Algorytm Grahama jest zazwyczaj około dwukrotnie szybszy.

Dla drugiego zestawu różnice były znaczące, dla zbioru zawierającego 100 punktów algorytm Grahama był 14 razy szybszy, a przy wzrastaniu liczby punktów różnica szybkości rosła jeszcze bardziej. Dla 1000 algorytm Grahama był już 671 razy szybszy. Było to spowodowane dużą ilością punktów należących do otoczki, dzięki temu złożoność algorytmu Jarvisa zawsze wynosiła $O(n^2)$.

Dla trzeciego i czwartego zbioru punktów algorytm Jarvisa zgodnie z przewidywaniami okazał się być szybszy, ponieważ wykonywał on tylko kilka iteracji. Było to spowodowane małą ilością punktów należących do otoczki.

W trzecim zbiorze dla 100 punktów algorytm Jarvisa okazywał się 2 razy wolniejszy, natomiast wraz ze wzrostem liczby punktów algorytm Jarvisa okazywał się być szybszy. Dla 100 000 punktów był on 6,12 razy szybszy.

W czwartym zbiorze algorytm Grahama dorównywał algorytmowi Jarvisa tyko dla 100 punktów. Dla większej ilości punktów algorytm Jarvisa był znacznie szybszy. Dla 100 000 punktów był on 173,4 razy szybszy.

8. Wnioski:

Oba algorytmy poprawnie wyznaczają otoczkę. Złożoność czasowa algorytmu Grahama wynosi O(nlogn), przez co dla dowolnego zbioru punktów czas jej wykonywania jest zbliżony. Natomiast algorytm Jarvisa najlepiej sprawdza się, jeżeli liczba wierzchołków otoczki jest mała.

Dlatego algorytm Jarvisa może być stosowany jeżeli mamy pewność, że analizujemy duże zbiory posiadające mało wierzchołków w otoczce. Natomiast jeżeli nie mamy takiej pewności, to algorytm Grahama jest dużo szybszy oraz ma podobny czas działania niezależnie od zbioru punktów.