Algorytmy geometryczne - Otoczka wypukła

Mikołaj Gaweł

27 października 2025

Dane podstawowe

- **Imię i nazwisko:** Mikołaj Gaweł
- **Data:** 27 października 2025
- **Tytuł ćwiczenia:** Algorytmy geometryczne – Otoczka Wypukła
- **Numer ćwiczenia:** 2

1 Dane techniczne

- **Język programowania:** Python 3.13.5
- **Środowisko uruchomieniowe:** Jupyter Notebook
- **Biblioteki:** `NumPy`, `Pandas`, standardowe biblioteki wbudowane w Pythona, oraz narzędzie do wizualizacji utworzone przez koło naukowe BIT.
- **System operacyjny:** Ubuntu 24.04 LTS, jądro Linux 6.14.0-33-generic
- **Architektura komputera:** 64-bitowa
- **Procesor:** AMD Ryzen 7 7735HS with Radeon Graphics (8 rdzeni / 16 wątków, max 4.83 GHz)
- **Pamięć RAM:** 64 GB

2 Wstęp teoretyczny

2.1 Otoczka wypukła

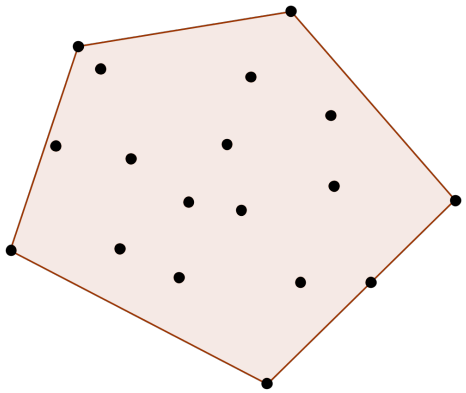
Niech Q będzie zbiorem punktów na płaszczyźnie $\mathbb{R}^2$. Otoczką wypukłą zbioru Q, oznaczaną $\mathrm{CH}(Q)$, nazywamy najmniejszy zbiór wypukły zawierający Q. Formalnie:

$$\mathrm{CH}(Q) = \bigcap \{C \subseteq \mathbb{R}^2 \mid C \text{ jest wypukły i } Q \subseteq C\}.$$

Intuicyjnie: otoczka wypukła to kształt jaki otrzymamy, naciągając elastyczną taśmę wokół punktów z Q.

Właściwości:

- Dla dowolnych dwóch punktów $p, q \in \mathrm{CH}(Q)$ odcinek $[p, q]$ leży w $\mathrm{CH}(Q)$.
- Otoczka wypukła jest wielokątem wypukłym; w praktyce reprezentujemy ją jako uporządkowaną sekwencję wierzchołków $v_1, v_2, \ldots, v_h$.



Rysunek 1: Przykładowa otoczka wypukła (punkty na czerwonej łamanej zamkniętej).

2.2 Podstawowe pojęcia pomocnicze

Dla trzech punktów $a = (a_x, a_y)$, $b = (b_x, b_y)$, $c = (c_x, c_y)$ definiujemy orientację (zwrot) trzech punktów przez wartość wyznacznika:

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

Interpretacja:

- $\mathrm{orient}(a, b, c) > 0$ — punkt c leży po lewej stronie wektora $a \to b$ (skręt *w lewo*),
- $\mathrm{orient}(a, b, c) < 0$ — punkt c leży po prawej stronie wektora $a \to b$ (skręt *w prawo*),
- $\mathrm{orient}(a, b, c) = 0$ — punkty są współliniowe.

Dla porównań odległości używamy kwadratu odległości euklidesowej (bez pierwiastka), aby uniknąć kosztownego obliczania $\sqrt{\cdot}$:

$$\mathrm{dist}^2(p, q) = (p_x - q_x)^2 + (p_y - q_y)^2.$$

2.3 Algorytm Grahama

Algorytm Grahama to klasyczna metoda wyznaczania otoczki wypukłej oparta na sortowaniu punktów względem kąta biegunowego względem wybranego punktu startowego.

Idea algorytmu

1. Wybierz punkt startowy s o najmniejszej współrzędnej y (przy remisie najmniejsza współrzędna x). Punkt s na pewno należy do otoczki.

2. Posortuj pozostałe punkty względem kąta biegunowego θ między osią x a wektorem $s \to p$. Przy współliniowości zachowaj punkty w porządku rosnącej odległości od s (bliższy pierwszy).

3. Iteracyjnie przeglądaj posortowaną listę i utrzymuj stos wierzchołków otoczki. Dla nowego punktu p dopóki ostatnie dwa punkty na stosie oraz p tworzą skręt w prawo (lub są współliniowe i punkt wewnętrzny), zdejmij wierzchołek ze stosu. Następnie włóż p na stos.

Algorytm 1 Algorytm Grahama

Dane wejściowe: Zbiór punktów Q
Dane wyjściowe: Lista wierzchołków otoczki wypukłej w porządku przeciwnym do ruchu wskazówek zegara

Wybierz $s = \arg\min_{p \in Q}(p_y, p_x)$
Posortuj $Q \setminus \{s\}$ według kąta względem s
Inicjuj pusty stos S; wstaw s oraz dwa pierwsze punkty
for każdy następny punkt p w porządku posortowanym **do**
    while rozmiar $S \geq 2$ **oraz** orient$(S[-2], S[-1], p) \leq 0$ **do**
        Usuń wierzchołek ze stosu S
    end while
    Dodaj punkt p do stosu S
end for
return Kolejność punktów w S

Złożoność

- Sortowanie — $O(n \log n)$.
- Przejście po posortowanej liście ze stosowaniem operacji push/pop — $O(n)$.

Łączna złożoność czasowa: $\mathbf{O(n \log n)}$. Złożoność pamięciowa: $\mathbf{O(n)}$.

Uwagi implementacyjne

- Trzeba ostrożnie traktować punkty współliniowe — decyzja, czy utrzymać skrajne punkty, decyduje o tym, czy otoczka zawiera wewnętrzne punkty na krawędzi.
- Przy sortowaniu bez użycia funkcji trygonometrycznych można porównywać punkty używając orientacji (porównania kątowe) i odległości.

2.4 Algorytm Jarvisa

Algorytm Jarvisa, zwany również gift wrapping, konstruuje otoczkę wypukłą przez „owijanie" punktów zewnętrznych.

Idea algorytmu

1. Rozpocznij od punktu o najmniejszej współrzędnej y (przy remisie najmniejsza współrzędna x). Niech będzie to punkt s, punkt s na pewno należy do otoczki.

2. Znajdź punkt q taki, że dla dowolnego innego punktu r punkt r leży po lewej stronie wektora $p \to q$ (czyli q jest najbardziej „prawy" względem p). W praktyce iterujemy po wszystkich punktach i wybieramy ten, dla którego orientacja jest najbardziej korzystna, p to punkt na otoczce obecnie przetwarzany

3. Ustaw $p := q$ i powtarzaj, aż wrócisz do punktu startowego, otrzyammy tak otoczkę zgodnie z kolejnością CCW

Algorytm 2 Jarvis

Dane wejściowe: zbiór punktów Q
Dane wyjściowe: lista wierzchołków otoczki wypukłej

```
wybierz p_0 = arg min_{p∈Q} p_x
p ← p_0
inicjalizuj pustą listę otoczki
while p ≠ p_0 lub otoczka jest pusta do
    wybierz q dowolny punkt różny od p
    for każdy punkt r ∈ Q do
        if orient(p, q, r) < 0 then
            q ← r
        else if orient(p, q, r) = 0 and dist^2(p, r) > dist^2(p, q) then
            q ← r
        end if
    end for
    dodaj q do otoczki
    p ← q
end while
return lista wierzchołków
```

Złożoność

- W kroku znajdowania następnego punktu wykonujemy $O(n)$ porównań; liczba iteracji równa jest liczbie wierzchołków otoczki h.

- Zatem czas całkowity: $\mathbf{O(n \cdot h)}$. W najgorszym przypadku, gdy $h = \Theta(n)$ (np. punkty na okręgu), złożoność to $\mathbf{O(n^2)}$.

Uwagi

- Jarvis jest prosty i wygodny dla przypadków, gdzie h jest małe (np. większość punktów jest w środku), ponieważ wtedy koszt jest bliski liniowemu.
- Dla punktów leżących na okręgu (najgorszy przypadek) Jarvis jest znacznie wolniejszy od Grahama.
- Tak jak w Grahamie, należy wziąć pod uwagę sytuacje współliniowe oraz duplikaty punktów.

3 Generowanie zbiorów i wizualizacja

W ćwiczeniu przygotowano cztery zbiory punktów w przestrzeni 2D (typ danych: `double`). Do generowania danych wykorzystano bibliotekę `NumPy`, a dla zachowania powtarzalności wyników ustawiono stałe ziarno generatora (`np.random.seed`).

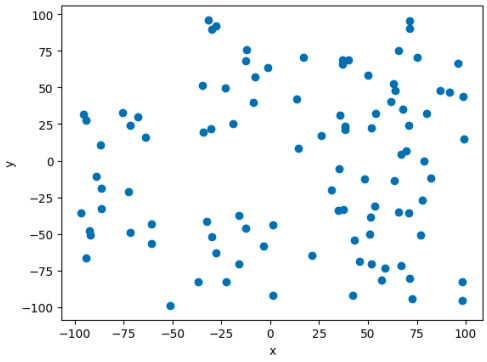
1. **Zbiór A:** 100 losowych punktów o współrzędnych (x, y) z przedziału $[-100, 100]$, wygenerowanych funkcją `np.random.uniform`.
2. **Zbiór B:** 100 losowych punktów leżących na okręgu o środku $(0, 0)$ i promieniu $R = 10$. Współrzędne punktów wygenerowano według równania parametrycznego:

   $$x = R\cos(\theta), \quad y = R\sin(\theta), \quad \theta \in [0, 2\pi]$$

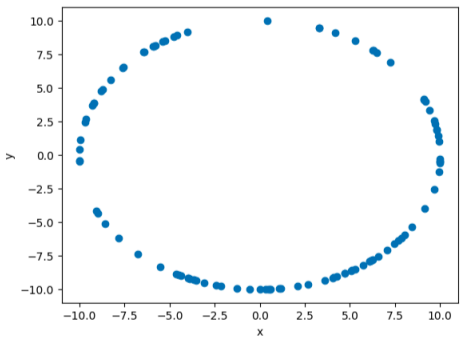
   gdzie każdy punkt ma losowo wybrany kąt θ z przedziału $[0, 2\pi]$.
3. **Zbiór C:** 100 losowych punktów leżących na obwodzie prostokąta o wierzchołkach $(-10, 10)$, $(-10, -10)$, $(10, -10)$, $(10, 10)$. Punkty wygenerowano w sposób równomierny wzdłuż obwodu: najpierw losowano bok z prawdopodobieństwem proporcjonalnym do jego długości, a następnie generowano punkt na tym boku parametrycznie:

   $$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad t \in [0, 1].$$

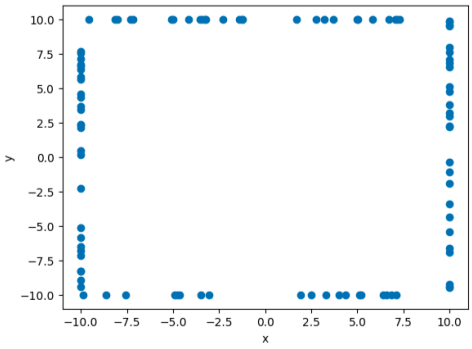
4. **Zbiór D:** zawiera wierzchołki kwadratu $(0, 0)$, $(10, 0)$, $(10, 10)$, $(0, 10)$ oraz dodatkowe punkty:
   - 25 punktów losowo wygenerowanych na każdym z dwóch boków kwadratu leżących na osiach,
   - 20 punktów losowo wygenerowanych na każdej przekątnej kwadratu.



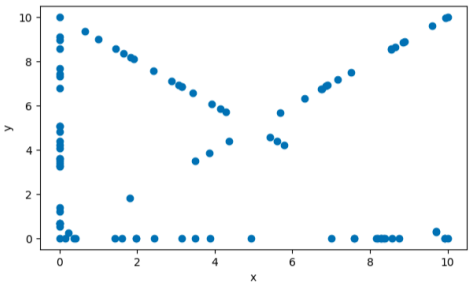
(a) Zbiór A: losowe punkty $[-100, 100]$



(b) Zbiór B: punkty na okręgu $R = 10$



(c) Zbiór C: punkty na bokach prostokąta



(d) Zbiór D: wierzchołki kwadratu oraz punkty losowe na bokach i przekątnych

Rysunek 2: Wizualizacja czterech zbiorów punktów wygenerowanych w przestrzeni 2D.

4 Realizacja algorytmów otoczki wypukłej oraz klasyfikacja ich działania

Ze względu na obliczenia na liczbach zmiennoprzecinkowych, w implementacji obu algorytmów (Grahama i Jarvisa) za tolerancję zera przyjęto stałą $\epsilon = 10^{-24}$. Wartość ta była używana do sprawdzania współliniowości punktów.

4.1 Implementacja algorytmu Grahama

Algorytm Grahama zaimplementowano w języku Python z wykorzystaniem funkcji `sorted()` i niestandardowego komparatora opartego na orientacji punktów. Do obliczania zwrotu trzech punktów użyto funkcji `orient(a,b,c)` zdefiniowanej jako wyznacznik:

$$\text{orient}(a, b, c) = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x).$$

Sortowanie punktów odbywało się względem kąta biegunowego bez użycia funkcji trygonometrycznych, wyłącznie na podstawie znaku orientacji względem punktu początkowego p_0. W przypadku współliniowości (gdy $|\text{orient}(p_0, a, b)| < 10^{-24}$) punkty porównywano względem ich odległości od p_0, zachowując bliższy punkt wcześniej. Po sortowaniu punkty były przeglądane kolejno, a na stosie utrzymywano kształt aktualnej otoczki. Punkty, które powodowały skręt w prawo lub współliniowość (orientacja $\leq 10^{-24}$), były usuwane ze stosu:

$$\text{while } \text{orient}(S[-2], S[-1], p) \leq 10^{-24} \text{ usuń } S[-1].$$

Dzięki temu punkty leżące wewnątrz krawędzi (współliniowe) były eliminowane z końcowego wyniku.

Podsumowanie implementacji Grahama:

- Sortowanie: według orientacji i odległości, bez funkcji trygonometrycznych.
- Kryterium eliminacji: punkty współliniowe wewnętrzne usuwane.
- Złożoność czasowa: $O(n \log n)$.

4.2 Implementacja algorytmu Jarvisa

Algorytm Jarvisa (*Gift Wrapping*) zrealizowano zgodnie z opisem teoretycznym. Startowy punkt p_0 to punkt o najmniejszej współrzędnej y, a przy remisie — o najmniejszej współrzędnej x. W każdej iteracji wybierano punkt q, który był najbardziej „prawy" względem bieżącego punktu p. W pętli iteracyjnej, jeśli testowany punkt r znajdował się na prawo od wektora $p \to q$ (tzn. $\text{orient}(p, q, r) < 0$), to r stawał się nowym kandydatem. Dodatkowo, jeśli punkty były współliniowe ($|\,\text{orient}\,| \leq 10^{-24}$), wybierano ten, który był dalej od p, aby uniknąć włączenia punktów wewnętrznych na tej samej linii.

Podsumowanie implementacji Jarvisa:

- Porównywanie punktów: na podstawie wyznacznika orientacji.
- Punkty współliniowe: utrzymywano skrajny (najdalszy) punkt.
- Złożoność czasowa: $O(nh)$, gdzie h to liczba punktów na otoczce.

4.3 Porównanie wyników dla zbiorów A–D

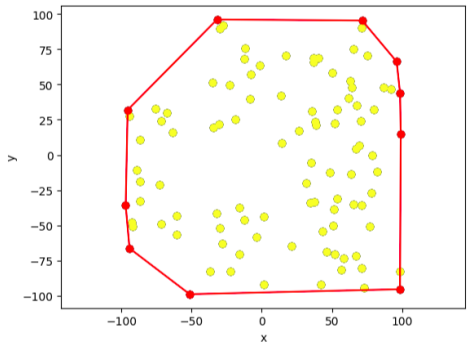
Dla każdego zbioru danych (Zbiór A–Zbiór D) oba algorytmy wyznaczyły identyczne zestawy punktów należących do otoczki wypukłej, co potwierdza poprawność implementacji. Liczba punktów na otoczce była taka sama dla Grahama i Jarvisa:

Zbiór	**Liczba punktów otoczki**	**Algorytmy (Graham / Jarvis)**
Zbiór A	10	10
Zbiór B	100	100
Zbiór C	8	8
Zbiór D	4	4

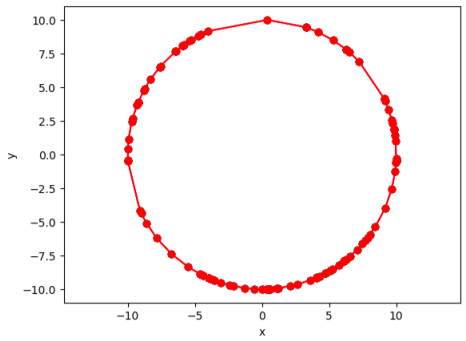
Warto zauważyć kluczową różnicę między Zbiorami C i D. Zbiór C (losowe punkty na bokach) nie zawierał wierzchołków, przez co otoczka została poprawnie wyznaczona przez 8 punktów (po dwa skrajne punkty przy każdym z czterech rogów). Zbiór D natomiast celowo zawierał 4 wierzchołki kwadratu, które (dzięki poprawnej obsłudze współliniowości) zostały wyznaczone jako jedyne punkty otoczki.

4.4 Wizualizacja otoczek wypukłych

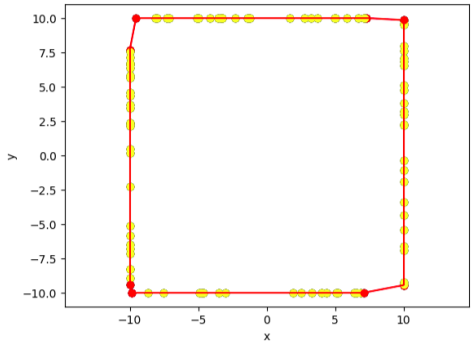
Na załączonych obrazach punkty należące do otoczki wypukłej zaznaczono kolorem czerwonym, a punkty, które do niej nie należą – kolorem żółtym. Sama otoczka jest przedstawiona jako czerwona łamana zamknięta.



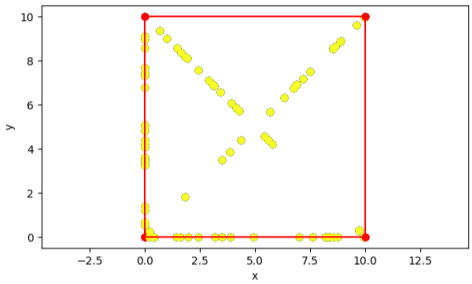
(a) Zbiór A — otoczka wypukła



(b) Zbiór B — otoczka wypukła



(c) Zbiór C — otoczka wypukła



(d) Zbiór D — otoczka wypukła

Rysunek 3: Otoczki wypukłe wyznaczone algorytmami Grahama i Jarvisa (rezultaty identyczne).

4.5 Analiza czasu działania

Dla każdego zbioru (Zbiór A–Zbiór D) wykonano pomiary czasu działania algorytmów Grahama i Jarvisa dla różnych rozmiarów danych ($n = 50, 100, 500, 1000, 5000, 25000$). Wyniki zestawiono w tabeli:

Zbiór	n	Punkty otoczki	Czas Grahama [s]	Czas Jarvisa [s]
A	50	11	0.00034	0.00058
A	100	12	0.00061	0.00111
A	500	14	0.00410	0.00576
A	1000	14	0.00807	0.00953
A	5000	25	0.04654	0.07551
A	25000	25	0.21769	0.42516
B	50	50	0.00017	0.00132
B	100	100	0.00032	0.00561
B	500	500	0.00225	0.17075
B	1000	1000	0.00642	0.60165
B	5000	5000	0.03028	13.81691
B	25000	25000	0.21531	396.47662
C	50	8	0.00029	0.00028
C	100	8	0.00072	0.00070
C	500	8	0.00495	0.00395
C	1000	8	0.01240	0.00677
C	5000	8	0.07221	0.03278
C	25000	8	0.47013	0.16191
D	50	4	0.00015	0.00008
D	100	4	0.00038	0.00016
D	500	4	0.00350	0.00083
D	1000	4	0.00592	0.00149
D	5000	4	0.03840	0.00842
D	25000	4	0.24384	0.04449

Tabela 1: Porównanie czasów działania algorytmów Grahama i Jarvisa dla zbiorów A–D.

5 Wnioski

- **Poprawność:** Obie implementacje zwróciły identyczne otoczki dla wszystkich zbiorów, co potwierdza ich poprawność i spójną obsługę przypadków współliniowych.
- **Wydajność (Złożoność):** Dane dla zbioru B (okrąg, $h = n$) idealnie potwierdzają teorię. Algorytm Grahama ($O(n \log n)$, 0.215s dla $n = 25000$) jest drastycznie wydajniejszy od Jarvisa, który osiąga pesymistyczną złożoność $O(n^2)$ (396s).
- **Przypadki szczególne:** Odwrotnie jest dla zbiorów z małym h (C i D), gdzie Jarvis ($O(nh)$) jest wyraźnie szybszy (dla D, $n = 25000$: 0.044s) niż Graham ($O(n \log n)$, 0.243s), gdyż jego koszt pokonuje sortowanie.
- **Rekomendacja praktyczna:** Algorytm Grahama ($O(n \log n)$) jest preferowanym wyborem dla ogólnych zastosowań. Jarvis ($O(nh)$) jest lepszy tylko wtedy, gdy istnieje pewność, że otoczka będzie miała bardzo mało wierzchołków ($h \ll \log n$).