Algorytmy geometryczne

Sprawozdanie z ćwiczenia 2.

Krzysztof Kwiecień
gr. Wt_14.40_B

Dane techniczne urządzenia na którym wykonano ćwiczenie:

Komputer z systemem Windows 10 x64
Procesor: Intel Core I5 2500k @4.5Ghz
Pamięć RAM: 8GB
Środowisko: Jupyter notebook

Ćwiczenie zrealizowano w języku Python 3, z wykorzystaniem bibliotek *numpy* (umożliwiających wiele operacji numerycznych), *random* (do generacji losowych punktów) oraz *matplotlib* (do rysowania wykresów).

Opis realizacji ćwiczenia:

Ćwiczenie polegało na implementacji oraz testach algorytmów Grahama oraz Jarvisa, wyznaczających otoczkę wypukłą zbioru punktów..

1. Generacja punktów

W celu wykonania ćwiczenia wygenerowane zostały 4 zbiory punktów (2D, typu double), które zostały umieszczone w osobnych tablicach:

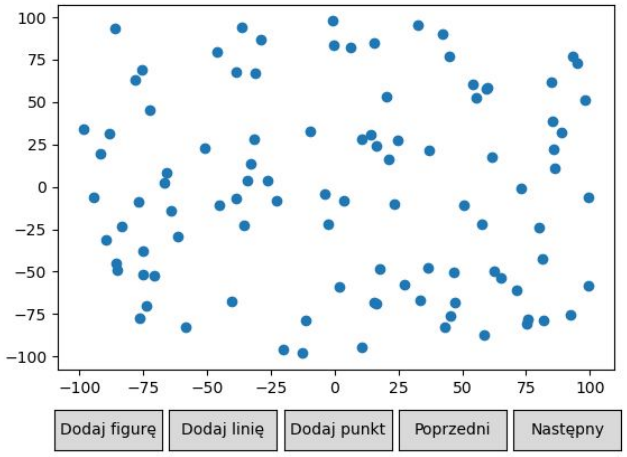
- Zestaw 1.
  100 losowych punktów o współrzędnych z przedziału [-100, 100]
- Zestaw 2.
  100 losowych punktów leżących na okręgu o środku (0,0) i promieniu R=10,
- Zestaw 3.
  100 losowych punktów leżących na bokach prostokąta o wierzchołkach (-10, 10), (-10,-10), (10,-10), (10,10)
- Zestaw 4.
  zawierający 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.

W celu porównania czasów wykonania algorytmów w ostatniej części ćwiczenia, dodatkowo zostały wylosowane zbiory o takich samych parametrach, jednakże z większymi liczbami punktów w każdym zestawie.

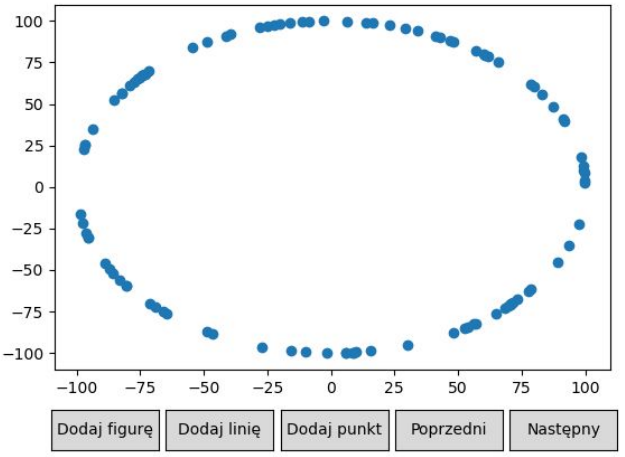
Losowanie położenia punktów odbyło się za pomocą metody random.uniform z biblioteki random. Funkcja ta generuje liczby typu double dla zadanego zakresu obustronnie domkniętego.
Punkty z zestawu 2. zostały wygenerowane z wykorzystaniem funkcji trygonometrycznych z biblioteki numpy.
Dla punktów z zestawu 3 wylosowano bok na którym mają zostać ułożone, a następnie konkretne współrzędne na tym boku.
Do wygenerowania punktów z zestawu 4. użyto równania prostej w celu ułożenia punktów ich na przekątnych kwadratu.

Wszystkie zestawy punktów zostały zwizualizowane za pomocą dostarczonego narzędzia graficznego opartego o bibliotekę *matplotlib*.

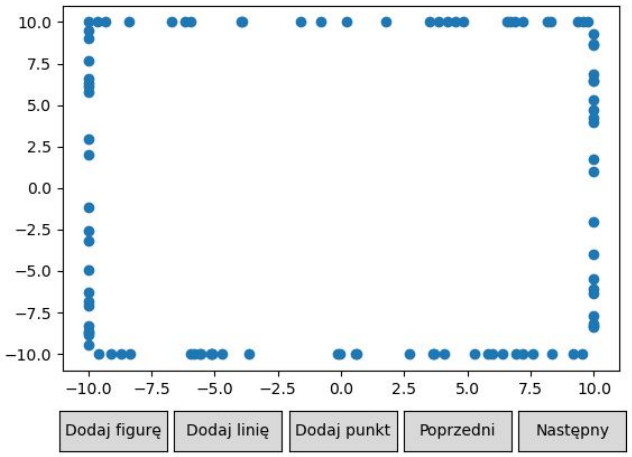
Wykres 1.1 Zestaw danych 1.



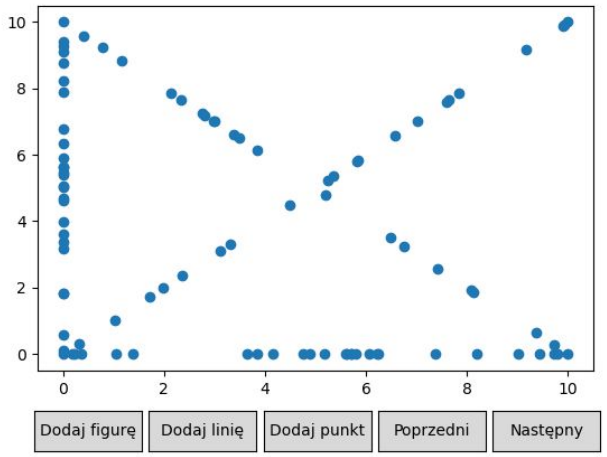
Wykres 1.2 Zestaw danych 2.



Wykres 1.3 Zestaw danych 3.



Wykres 1.4 Zestaw danych 4.



2. Metoda obliczania wyznacznika oraz tolerancja dla zera

Do tego ćwiczenia wykorzystano wyznacznik 3x3 własnej implementacji. Jako tolerancję dla zera przy określaniu orientacji punktu względem odcinka przyjęto 10^{-12}. Dla niższej tolerancji, na poziomie 10^{-14} algorytm Jarvisa czasami błędnie wyznaczał otoczkę w zestawie 2.

3. Implementacja algorytmu Grahama

Algorytm działa w następujący sposób: Na początku znajduje punkt o najmniejszej współrzędnej y, jeśli takich punktów jest kilka, wybiera ten o najmniejszej współrzędnej x. Jest to punkt startowy. Następnie dokonuje sortowania punktów z wykorzystaniem algorytmu quicksort. Punkty sortowane są wg. kąta jaki tworzy wektor punktu startowego z rozważanym względem dodatniego kierunku osi x. W przypadku gdy rozpatrywane punkty tworzą taki sam kąt, pierwszeństwo mają punkty bardziej oddalone od punktu startowego.
W głównej pętli algorytmu najpierw sprawdzamy, czy rozpatrywany punkt jest współliniowy z ostatnią krawędzią otoczki, jeśli tak, to aktualizujemy ją tak aby rozszerzyć ją do tego punktu. Robimy to ponieważ podczas sortowania nie zostały usunięte punkty współliniowe w celu możliwości pracy na oryginalnym zbiorze. Następnie standardowo sprawdzamy orientację punktu względem ostatniej krawędzi i dokładamy punkty do otoczki bądź cofamy się. Po wykonaniu głównej pętli sprawdzamy czy ostatni punkt nie jest zbędny (współliniowy z punktem startowym i przedostatnim otoczki).

4. Implementacja algorytmu Jarvisa

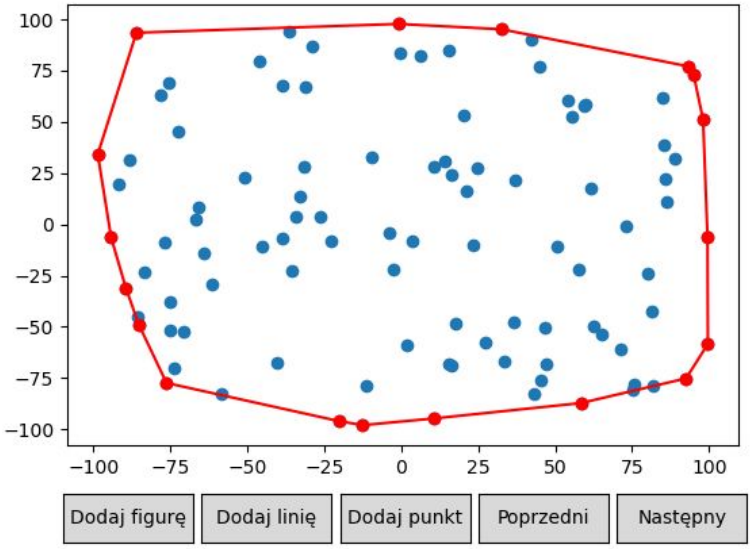
Podobnie jak w przypadku poprzedniego algorytmu na początku znaleziono punkt startowy. W wewnętrznej pętli algorytmu poszukiwano punktu, dla którego wszystkie pozostałe leżą po lewej stronie odcinka wyznaczonego przez poprzedni punkt otoczki i tenże punkt. W przypadku znalezienia kilku współliniowych punktów spełniających to założenie wybierano ten najbardziej oddalony od ostatniego punktu otoczki (tak aby nowo powstała krawędź była jak najdłuższa). Znaleziony punkt dodawano do otoczki i kontynuowano do momentu dotarcia z powrotem do punktu startowego.

5. Wyniki działania algorytmów Grahama i Jarvisa

Oba algorytmy przetestowano na tych samych zbiorach danych. W przypadku obu algorytmów uzyskane zbiory punktów otoczki wypukłej były poprawne, dla każdego zbioru takie same dla obydwu algorytmów.

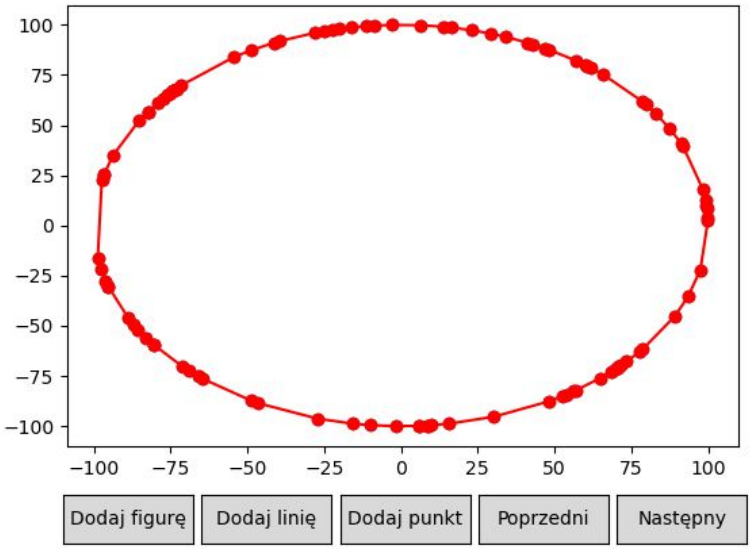
Pierwszy zbiór był z punktu widzenia obu algorytmów dość prostym przypadkiem - wystąpiła w nim bardzo mała ilość punktów współliniowych leżących na krawędziach otoczki, zostały one poprawnie nie zawarte w wynikowym zbiorze punktów.

Wykres 2.1 Otoczka wypukła 1. zestawu danych wygenerowana przez algorytmy Grahama i Jarvisa



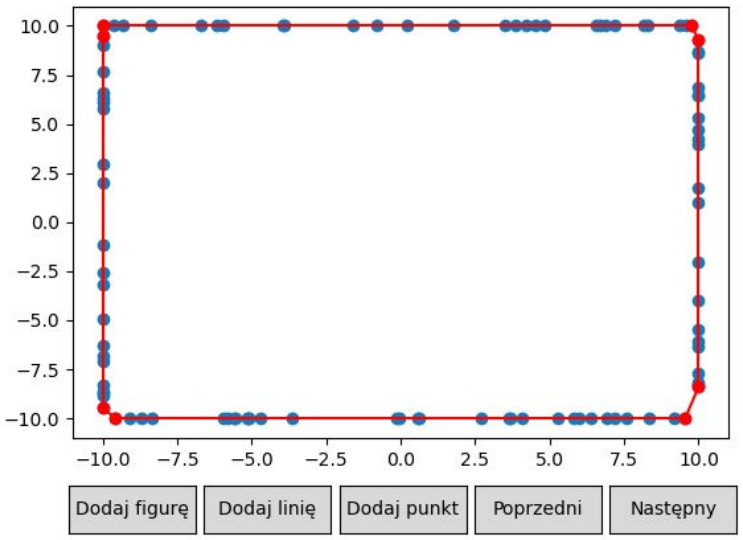
Drugi zbiór ukazał znaczącą przewagę algorytmu Grahama nad algorytmem Jarvisa. Dla obu algorytmów została wyznaczona ta sama otoczka, jednakże algorytm Grahama zrobił to odczuwalnie szybciej. Dużą wadą algorytmu Jarvisa jest konieczność sprawdzenia wielu punktów w celu wyznaczenia kolejnego punktu otoczki, nawet gdy między punktami panuje pewien porządek. Uporządkowanie punktów w tym zestawie sprawia, że algorytm Grahama bardzo szybko odnajduje poprawną otoczkę, ponieważ każdy następny rozpatrywany punkt będzie do tej otoczki należał. Na przykładzie tego zestawu (sposobu przebiegu algorytmu dla niego) widzimy, że sposób rozłożenia punktów na płaszczyźnie dla algorytmu Jarvisa nie ma znaczenia.

Wykres 2.2 Otoczka wypukła 2. zestawu danych wygenerowana przez algorytmy Grahama i Jarvisa



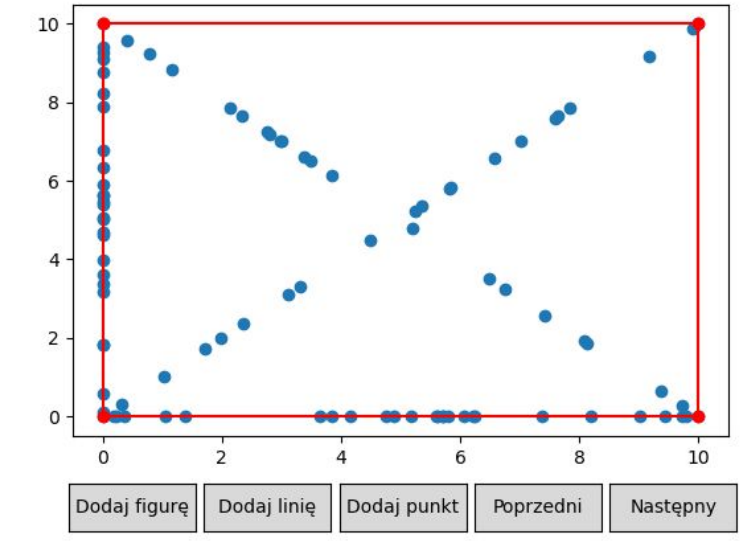
Trzeci zestaw danych testuje problem współliniowości w otoczce. Oba algorytmy radzą sobie z tym bez problemu dzięki dodatkowym instrukcjom dla wyboru punktów w tym przypadku. Powstała otoczka ma kształt oktagonu, czyli dla każdego boku prostokąta w otoczce znajdują się punkty skrajne tego boku. Teoretycznie możliwe jest, aby wylosowany został punkt umieszczony idealnie w rogu, lecz jest to praktycznie niemal niemożliwe, aczkolwiek niższa tolerancja mogłaby tak sklasyfikować wystarczająco bliski punkt, w wyniku czego otoczka mogłaby mieć od czterech do ośmiu boków.

Wykres 2.3 Otoczka wypukła 3. zestawu danych wygenerowana przez algorytmy Grahama i Jarvisa



Ostatni zestaw również testował problem współliniowości, lecz także ekstremalny przypadek “cofania się” algorytmu Grahama. Dzięki zastosowanej metodzie sortowania (z uwzględnieniem odległości) nie wystąpiły żadne problemy. Jako iż w mojej implementacji punkty współliniowe nie są usuwane ze zbioru punktów, w głównej pętli sprawdzamy większość punktów leżących na przekątnych, czym alternatywnie moglibyśmy się zająć wcześniej (usunąć te punkty ze zbioru wejściowego). Co ciekawe, zestaw ten jest przykładem jednej z niewielu sytuacji, w której algorytm Jarvisa jest szybszy od algorytmu Grahama. Wynika to z wcześniej wspomnianego faktu konieczności sprawdzenia przez algorytm Grahama wszystkich punktów, podczas gdy w rzeczywistości występują jedynie 4 punkty, które algorytm Jarvisa musi znaleźć, aby zakończyć działanie. Innymi słowy dla tak mało skomplikowanej otoczki podejście losowe jest szybsze.
W obu przypadkach jest nią kwadrat, o wierzchołkach którymi został zdefiniowany zbiór punktów.

Wykres 2.4 Otoczka wypukła 4. zestawu danych wygenerowana przez algorytmy Grahama i Jarvisa



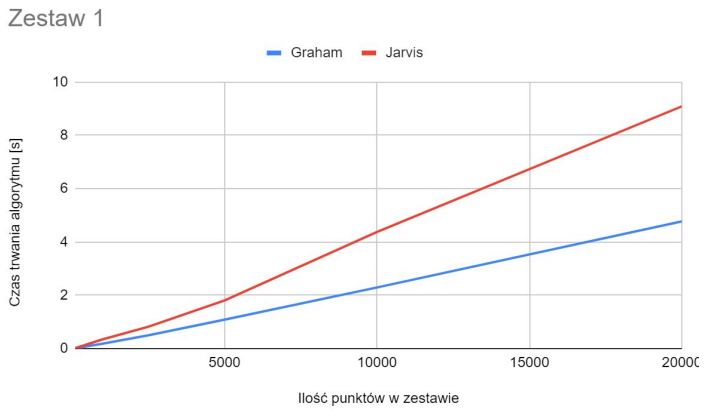
6. Porównanie czasu wykonania algorytmów Grahama i Jarvisa

W ostatniej części ćwiczenia algorytmy uruchomiono ponownie dla tych samych zestawów, w celu porównania ich czasu wykonania. W tym celu wyłączono w nich funkcjonalności związane z wizualizacją, aby uzyskane czasy były rzeczywistymi czasami wykonania samych algorytmów.
Testy wydajności przeprowadzono dla następujących ilości punktów z zestawach:
100, 1000, 2500, 5000, 10000, 2000.
Wyniki były zgodne z oczekiwaniami.
Dla pierwszego, w pełni losowego zbioru algorytm Grahama okazał się szybszy od Jarvisa, około dwukrotnie. Największa różnica wystąpiła dla zestawu drugiego. Już dla najmniejszej liczby punktów algorytm Grahama okazał się szybszy o aż 17.3x od algorytmu Jarvisa, co zostało spowodowane dużą liczbą punktów należących do otoczki. Bardzo dobrze widać tutaj kwadratową (w przypadku braku ograniczenia liczby wierzchołków otoczki) złożoność algorytmu Jarvisa. Dla 10x większej ilości punktów algorytm wykonuje się ok 100x dłużej, oraz dla zestawu większego 50x wykonuje się 2500x dłużej. W związku z naturą tej złożoności nie udało się przeprowadzić pomiaru dla algorytmu Jarvisa w zestawie 2. dla większej liczby punktów niż 5000 ze względów praktycznych, ponieważ algorytm wykonywałby się kilkanaście minut lub nawet kilkadziesiąt.
Zestaw 3. Dla większej liczby punktów wypadł na korzyść algorytmu Jarvisa. Spowodowane to było małym rozmiarem otoczki, dzięki czemu algorytm Jarvisa musiał znaleźć niedużą liczbę punktów aby zakończyć swoje działanie. Taka sama sytuacja miała w ostatnim zestawie, aczkolwiek w jego przypadku liczba punktów otoczki była tak mała, że algorytm Jarvisa działał wielokrotnie szybciej niż algorytm Grahama. Jak napisano wcześniej, wynika to z ograniczenia złożoności algorytmu Jarvisa przez liczbę wierzchołków w otoczce, co dla bardzo małej ilości wierzchołków (jak tylko 4 punkty w zestawie 4.) daje nam złożoność praktycznie liniową, co widzimy w tabeli 1.
Warto zwrócić uwagę na czas wykonania algorytmu Grahama pomiędzy zestawami - jest on bardzo zbliżony. Wynika to ze złożoności tego algorytmu - O(nlogn), a więc niezależnej od rodzaju/położenia punktów na płaszczyźnie, a jedynie od ich liczby.

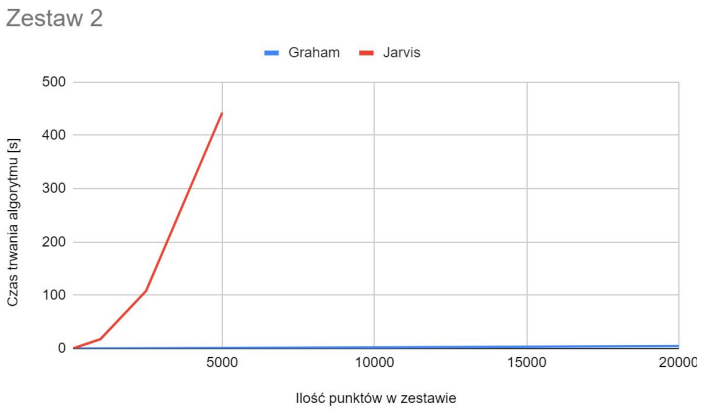
Tabela 1. Czas wykonywania algorytmów Grahama i Jarvisa dla poszczególnych zestawów w zależności od liczby punktów w zestawie (w sekundach)

Zestaw	Algorytm	100	1000	2500	5000	10000	20000
Zestaw 1	**Graham**	0.0110	0.1830	0.4975	1.0845	2.2920	4.7715
	Jarvis	0.0150	0.3445	0.8185	1.8050	4.3735	9.0880
Zestaw 2	**Graham**	0.0100	0.1860	0.4790	1.0545	2.1670	4.7045
	Jarvis	0.1730	17.5770	108.1330	442.6130	-	-
Zestaw 3	**Graham**	0.0090	0.1735	0.4590	1.1335	2.2835	4.9450
	Jarvis	0.0140	0.1415	0.3420	0.6920	1.3780	2.7175
Zestaw 4	**Graham**	0.0115	0.1895	0.5890	1.1275	2.7040	5.6835
	Jarvis	0.0065	0.0635	0.1610	0.3185	0.6230	1.2315

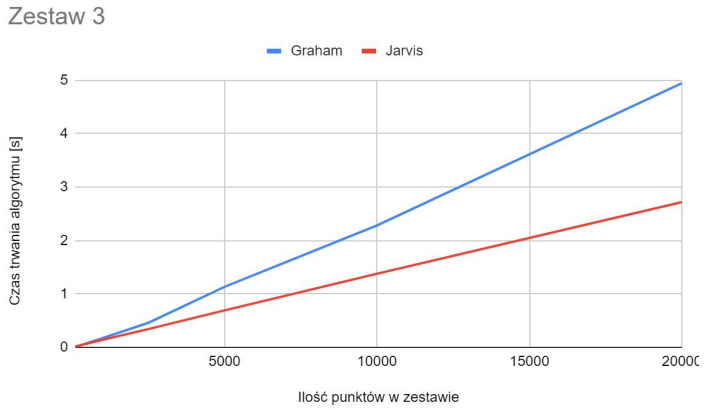
Wykres 3.1 Czas wykonania algorytmów Grahama i Jarvisa dla 1. zestawu danych w zależności od ilości punktów w zestawie



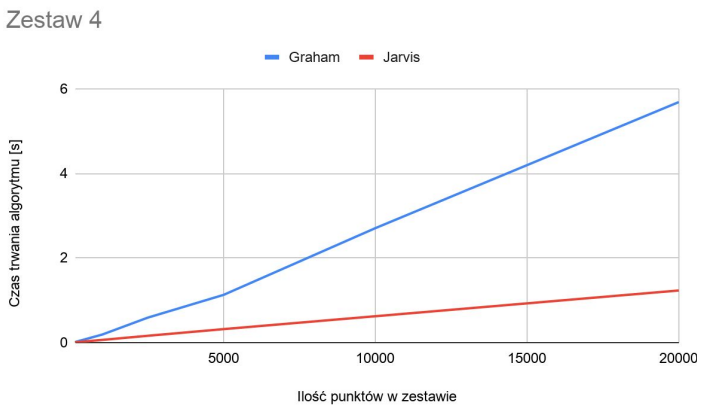
Wykres 3.2 Czas wykonania algorytmów Grahama i Jarvisa dla 2. zestawu danych w zależności od ilości punktów w zestawie



Wykres 3.3 Czas wykonania algorytmów Grahama i Jarvisa dla 3. zestawu danych w zależności od ilości punktów w zestawie



Wykres 3.4 Czas wykonania algorytmów Grahama i Jarvisa dla 4. zestawu danych w zależności od ilości punktów w zestawie



Wnioski i spostrzeżenia

Jak widać na podstawie powyższych danych oba algorytmy potrafią prawidłowo wyznaczyć otoczkę wypukłą dla dowolnych zbiorów danych, nawet takich, które z teoretycznego punktu widzenia mogłyby sprawiać trudności. Jak widzimy czas wykonania algorytmu Grahama dla różnych zbiorów o zbliżonej liczbie punktów jest bardzo podobny. W przypadku algorytmu Jarvisa różnice są kolosalne - jest to tłumaczone złożonością mającą związek z rozmiarem otoczki - rzędu O(kn) - gdzie k to ilość wierzchołków otoczki. Wyjaśnia to dobre czasy dla dwóch ostatnich zestawów danych i fatalnego czasu dla zestawu drugiego, w którym jako że otoczka zawiera wszystkie punkty mamy do czynienia z złożonością O(n^2), efekt ten jest zwłaszcza widoczny dla większej ilości punktów. Już dla 1000 w przypadku zestawu drugiego algorytm Jarvisa wykonywał się ok. 18 sekund, podczas gdy algorytm Grahama zrobił to w ułamku sekundy.

Algorytm Jarvisa może znaleźć zastosowanie w przypadku wyznaczania dużej ilości otoczek dla zbiorów, których otoczki będą posiadały mało wierzchołków. W innych przypadkach algorytm Grahama jest lepszym rozwiązaniem. Nie jest on bardziej skomplikowany ani trudniejszy w implementacji od algorytmu Jarvisa, wymaga jednak posiadania funkcji efektywnie sortującej punkty, daje on jednak znacznie lepszą złożoność, która pozwala wyznaczać otoczkę wypukłą dla dużych, skomplikowanych zbiorów punktów, a czas jego wykonania jest łatwo przewidywalny.