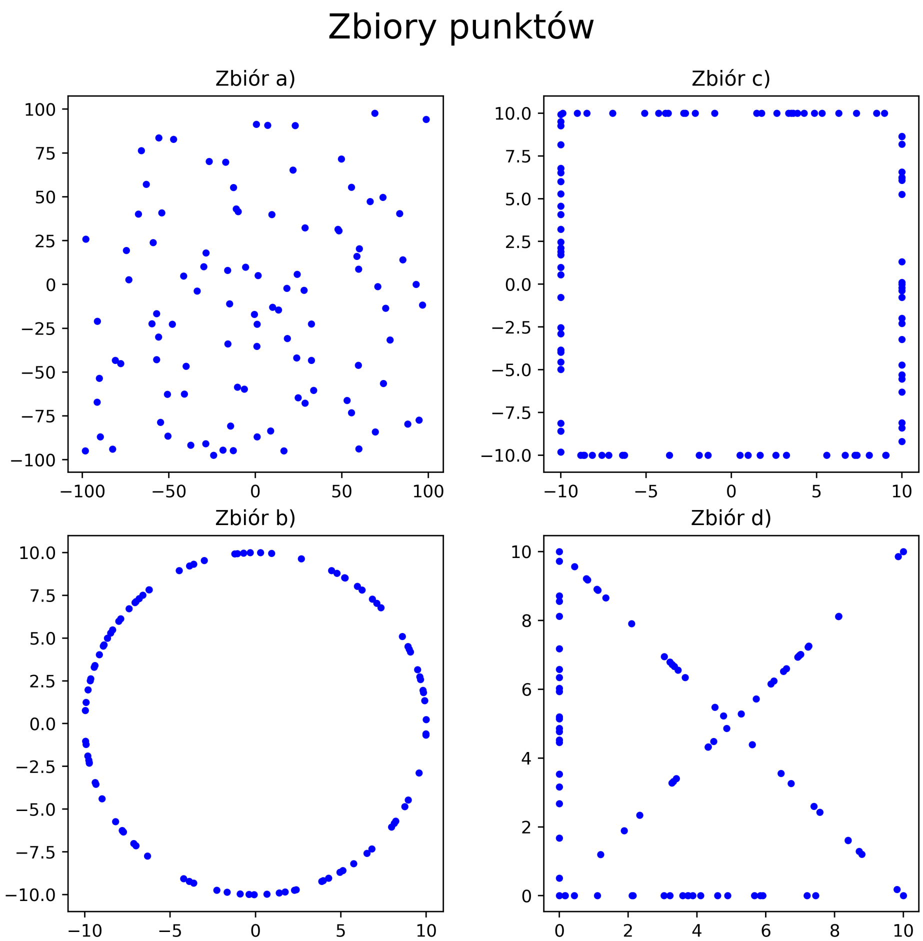
Opracowanie wyników laboratoriów

Generowanie punktów

Wszystkie operacje wykonane zostały na komputerze stacjonarnym z procesorem i5-7600k. Językiem z jakiego korzystałem był Python w wersji 3.10. Za pomocą biblioteki **NumPy** wygenerowałem 4 zadane zbiory punktów co zajęło około 0.001s:



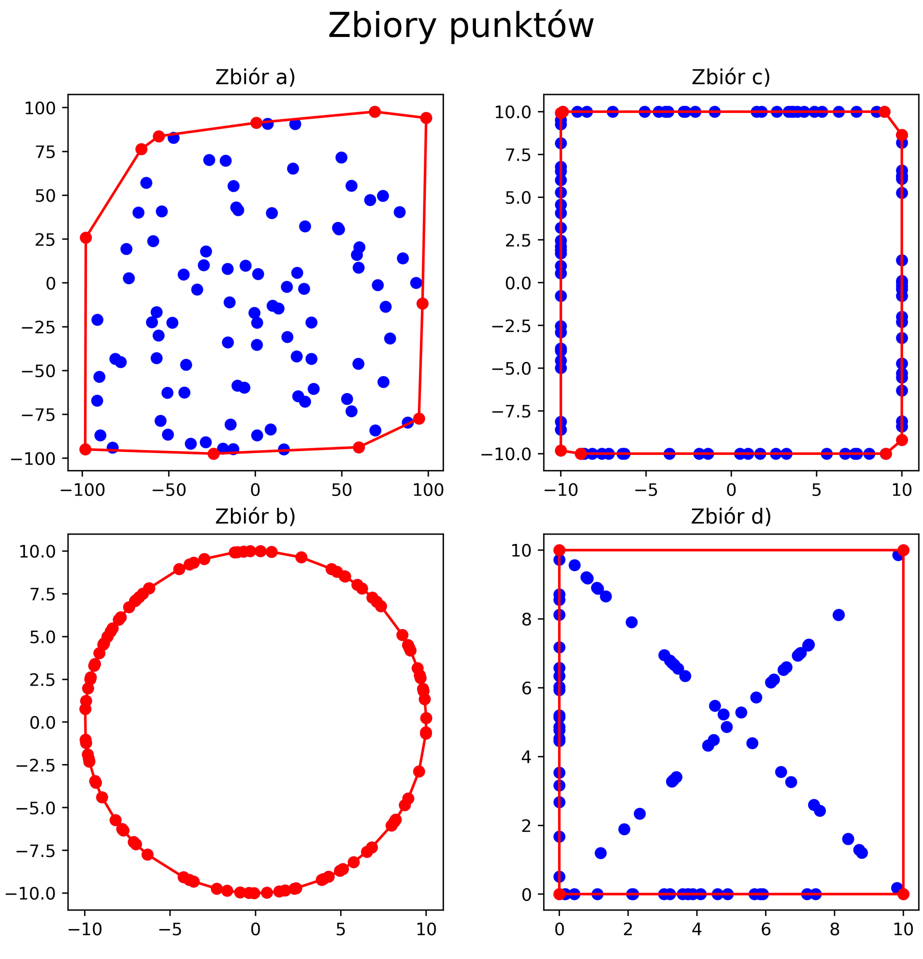
Wizualizacja 1: Wygenerowane zbiory punktów.

Wyznaczanie otoczki wypukłej

Algorytm Grahamsa oparłem głownie na funkcji `grahams_cmp(a,b,c)`, która jest komparatorem porównującym dwa elementy(`b,c`) z listy, w odniesieniu do `a`. Dzięki niej odbywa się sortowanie punktów po kącie (najmniejszym) jaki tworzą z wybranym punktem startowym, następnie po odległości (najmniejszej). Korzysta ona z zaimplementowanej przeze mnie funkcji `det` liczącej wyznacznik standardową metodą dla macierzy **2x2**. Następnie wykonywana jest pętla określona w algorytmie Grahamsa dodająca kolejne punkty do stosu i usuwająca te błędne.

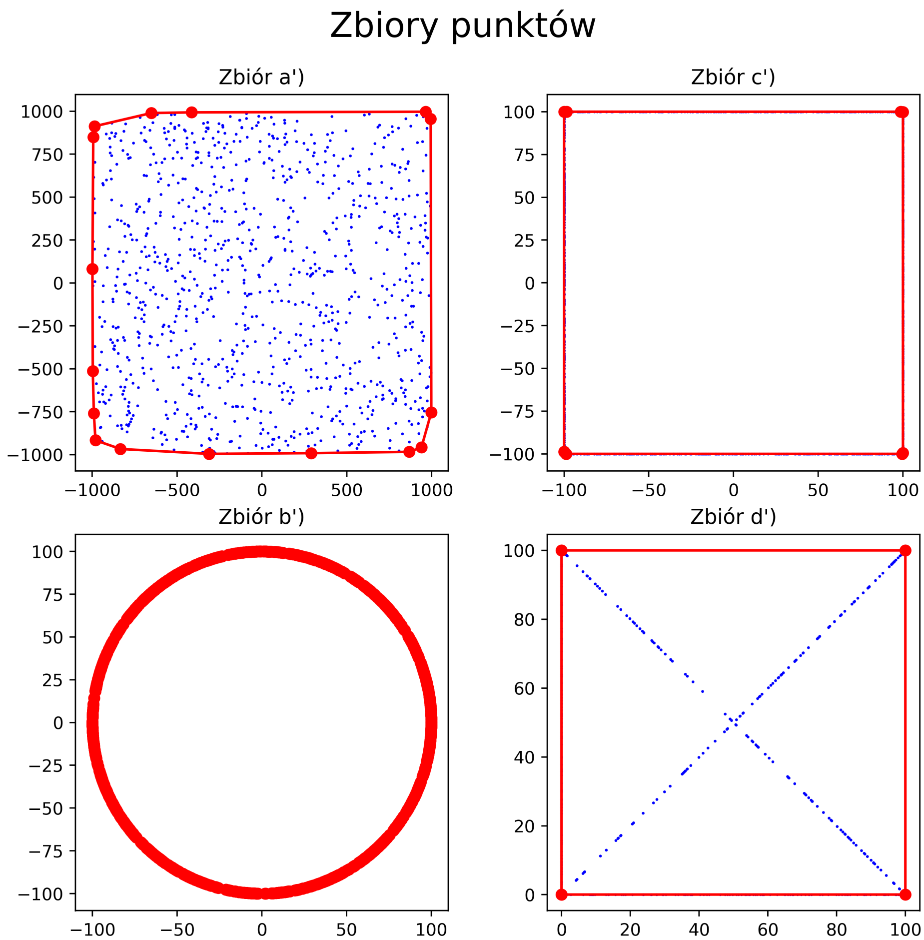
Algorytm Jarvisa oparłem na na funkcji `min_angle(a,b,c)`, która działa tak samo jak `grahams_cmp`, jednak różni się typem zwracanych danych, ponieważ zwraca ona jeden z `b,c`, który tworzy z `a` oraz OX najmniejszy kąt, a dla tego samego kąta, większą odległość. Przy użyciu `reduce` z **functools**, możemy przejść po wszystkich punktach wyciągając te, które z obecnie ostatnim, tworzą najmniejszy kąt. Będziemy to wykonywać, aż nie trafimy na punkt startowy.

Następnie używając wyżej wymienionych metod, wygenerowałem otoczki wypukłe uzyskanych wcześniej zbiorów punktów. Jako, że algorytmy posiadają złożoności: $n\log(n)$ - Grahamsa oraz nk - Jarvisa, gdzie $k =$ ilość punktów w otoczce, to dla otoczek zawierających mniej punktów, niż $log(n)$ algorytm Jarvisa powinien być szybszy. Wygenerowanie otoczki dla wszystkich zbiorów z punktu 1) zajęło odpowiednio 0.006s - Jarvis, 0.001s - Graham. W dalszej części przedstawie szczegółowe dane dla każdej z metod. Poniżej w *Wizualizacji 2.* zamieszczam ilustracje otoczek wypukłych dla zbiorów wygenerowanych wcześniej. Na czerwono zaznaczone są punkty należące do otoczki wraz z bokami wielokąta, który ją tworzy.



Wizualizacja 2: Otoczki wypukłe zbiorów z *wizualizacji 1.*

Poniżej w *wizualizacji 3.* przedstawiłem także otoczki wypukłe dla każdego ze zbiorów podobnych do tych z punktu pierwszego, jednak o liczności odpowiednio 1000, 1000, 1000 oraz 200.



Wizualizacja 3: Otoczki dla zbiorów o większej liczebności.

W programie dodałem także funkcje do wizualizacji algorytmu Grahamsa oraz Jarvisa. Są one dane odpowiednio `grahams_with_wiz` oraz `jaarvis_wth_wiz`. Zwracają one sceny interpretowane przez narzędzie wizualizacji graficznej.

Pomiar czasu

Jako, że zaimplementowane przeze mnie funkcje generujące zbiory pozwalają na modyfikowanie ich parametrów, pozwala to zmierzyć efektywność zaimplementowanych algorytmów. W tabelach poniżej zamieszczone zostały czasy potrzebne to wyznaczenia otoczki dla zbiorów o rożnej liczności. Czasy podane są dla algorytmów bez żadnych elementów tworzących wizualizacje oraz oparte na podobnych funkcjach pomocniczych, więc powinny one uwidaczniać prawdziwe różnice w złożonościach.

Tabela 1: Pomiar czasu dla zbiorów o rozkładzie jednolitym - jak a).

LICZNOŚĆ / **METODA**	10^3	10^4	10^5
Grahams	0.0089s	0.13s	1.84s
Jarvis	0.0079s	0.13s	1.98s

Tabela 2: Pomiar czasu dla zbioru punktów leżących na okręgu - jak b).

LICZNOŚĆ / **METODA**	10^2	10^3	10^4
Grahams	0.001s	0.005s	0.082s
Jarvis	0.005s	0.496s	50.79s

Tabela 3: Pomiar czasu dla zbioru punktów leżących na prostokącie - jak c).

LICZNOŚĆ / **METODA**	10^3	10^4	10^5
Grahams	0.003s	0.11s	0.67s
Jarvis	0.004s	0.051s	0.53s

Tabela 4: Pomiar czasu dla zbioru punktów bokach i przekątnych kwadratu - jak d).

LICZNOŚĆ / **METODA**	10^2	10^3	10^4	10^5
Grahams	0.001s	0.02s	0.30s	3.45s
Jarvis	0.001s	0.01s	0.09s	1.14s

Wnioski

Widzimy, że dla zbiorów, w których liczność punktów otoczki jest mała, 8 dla zbioru c) oraz 4 dla zbioru d), algorytm Jarvisa zaczyna być bardziej efektywny przy $k \ll n$. Natomiast dla zbiorów, gdzie liczność otoczki jest proporcjonalna do n, takich jak b), algorytm Jarvisa jest znacznie wolniejszy niż Grahama. Pozwala to postawić wniosek, że dla zbioru o nieznanym rozkładzie punktów warto zastosować algorytm o złożoności $n \log(n)$.