Algorytmy geometryczne
Sprawozdanie z laboratorium 2.

Michał Nożkiewicz
gr. Czw_11:20_B

1. Użyte narzędzia i oprogramowanie

Ćwiczenie realizowałem na komputerze z systemem Windows 10 x64**,**
Procesor: Intel Core i5-4460 @3.2 GHz
Pamięc RAM: 8GB

Kod programu pisałem w środowisku Jupyter notebook, w języku Python, z wykorzystaniem bibliotek *numpy, math, random, time* oraz *matplotlib.*

2. Opis ćwiczenia

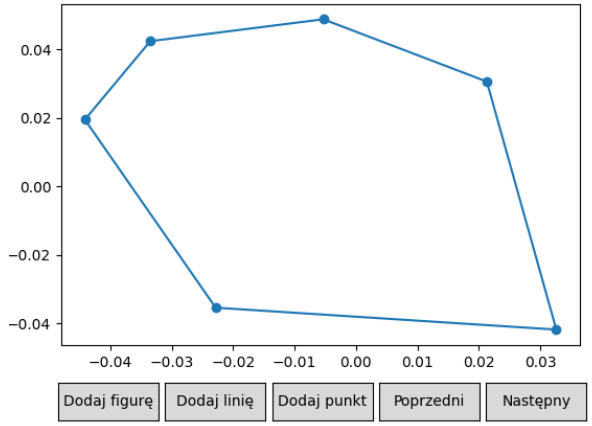
Zadaniem tego laboratorium było zaimplementowanie algorytmów sprawdzania czy dany wielokąt jest y-monotoniczny, klasyfikacji wierzchołków w dowolnym wielkokącie, a także triangulacji wielokąta y-monotonicznego.

3. Tworzenie zbiorów punktów

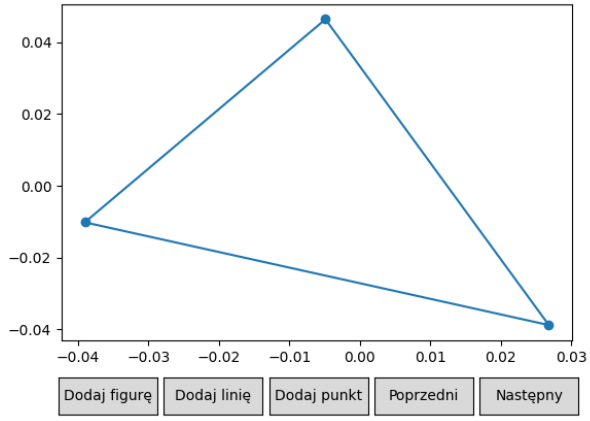
Podczas testowania algorytmów, korzystałem z własnoręcznie wprowadzonych zbiorów punktów, które następnie zapisywałem do pliku json. Przy wprowadzaniu figury przyjąłem założenie, że kolejne punkty dodaje w kolejności przeciwnej do ruchu wskazówek zegara. Wśród wybranych zbiorów punktów znalazły się figury proste do triangulacji takie jak wielokąty wypukłe, trójkąt, dla którego triangulacja nic nie powinna znajdować, a także figury które nie są y-monotoniczne. Pozostałe miały jak najbardziej utrudnione kształty.

Utworzyłem także funkcje, które, które przy wczytywaniu listy punktów, zwracały listę kolejnych boków wielokąta oraz listy, które dla każdego punktu przechowywały jego sąsiadów.

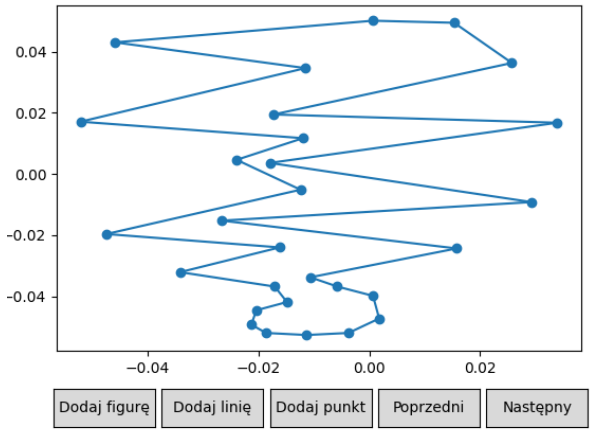
Wykres 3.1 Zbiór A



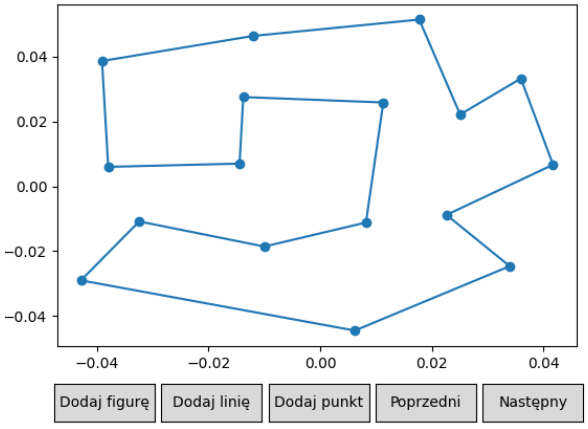
Wykres 3.2 Zbiór B



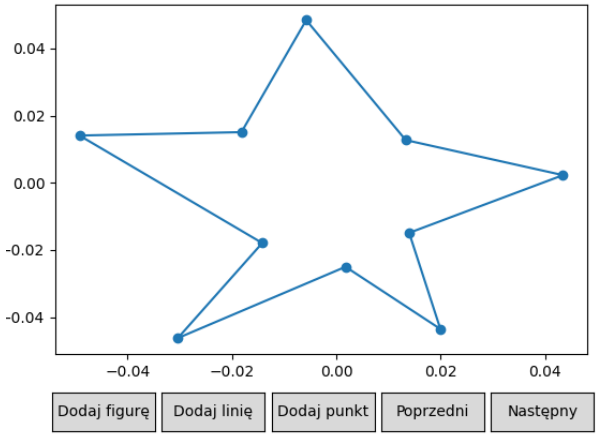
Wykres 3.3 Zbiór C



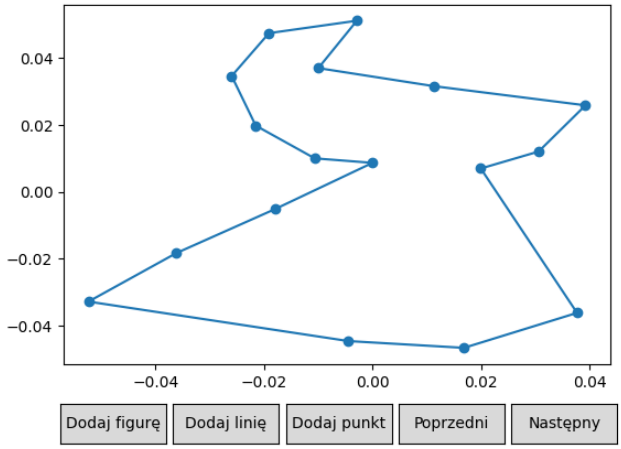
Wykres 3.1 Zbiór D



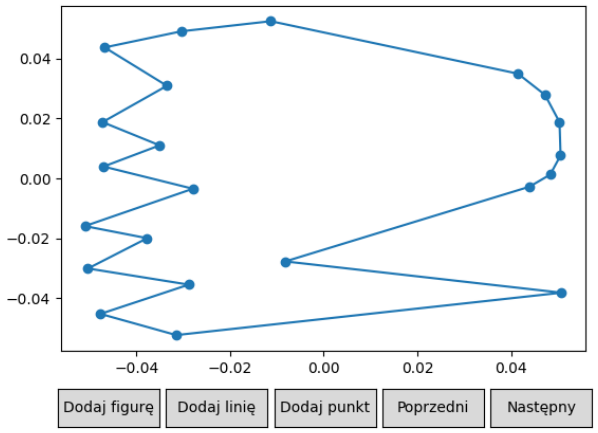
Wykres 3.1 Zbiór E



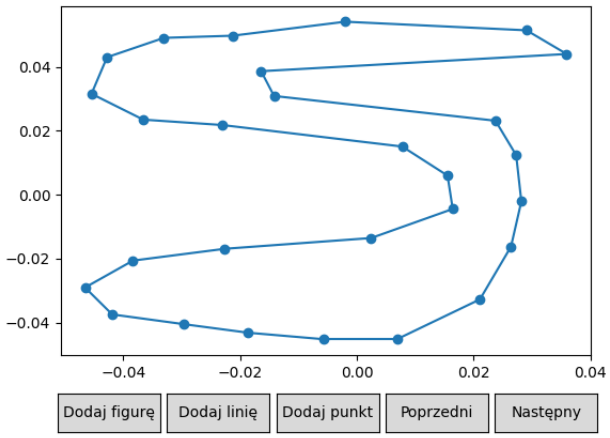
Wykres 3.1 Zbiór F



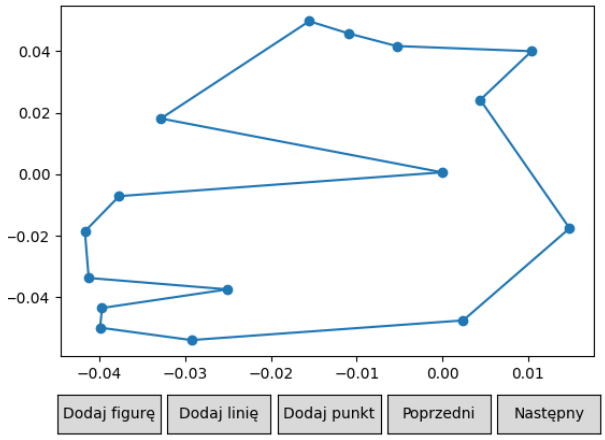
Wykres 3.1 Zbiór G



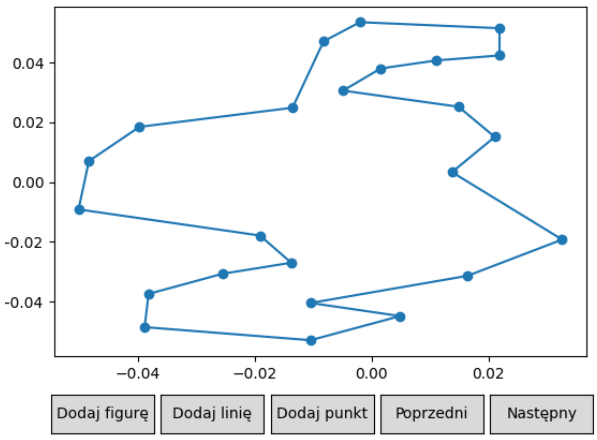
Wykres 3.1 Zbiór H



Wykres 3.1 Zbiór I



Wykres 3.1 Zbiór J



4. Sprawdzanie monotoniczności i klasyfikacja punktów

W wielokącie dokonujemy następujące podziału wierzchołków (w nawiasie kolory jakimi będę oznaczał poszczególne punkty):

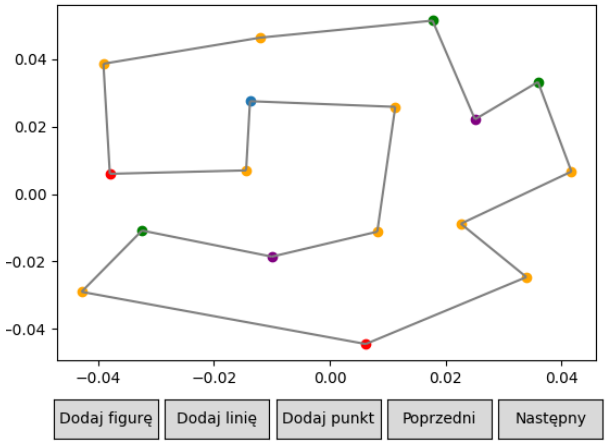
1. początkowy – obaj sąsiedzi leżą poniżej i kąt wewnętrzny < π(zielony),
2. końcowy – obaj sąsiedzi powyżej i kąt wewnętrzny < π(czerwony),
3. łączący – obaj sąsiedzi powyżej i kąt wewnętrzny > π(fioletowy),
4. dzielący – obaj sąsiedzi poniżej i kąt wewnętrzny > π(niebieski),
5. końcowy – jeden sąsiad wyżej, drugi niżej(pomarańczowy).

Wiadomo, że wielokąt monotoniczny nie może mieć wierzchołków dzielących ani łączących, więc algorytm sprawdzenia monotoniczności był bardzo podobny do

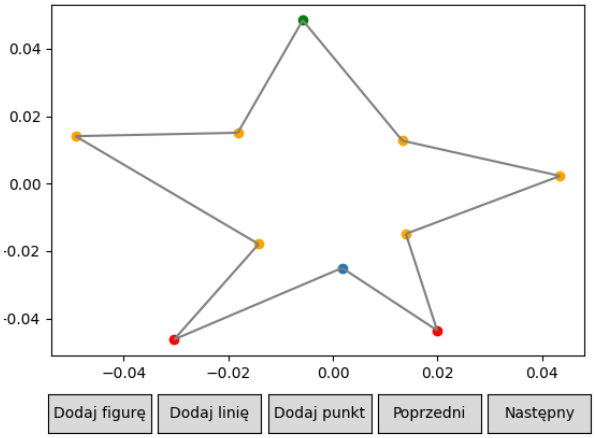
klasyfikacji punktów i polegał na jednokrotnym przejściu po liście punktów i sprawdzeniu położenia punktów względem jego sąsiadów. Kąt sprawdzałem używając wyznacznika. Można zauważyć, że jeśli idziemy przeciwnie z ruchem wskazówek zegara to wierzchołek o kącie mniejszym od π powiniem znaleźć się po prawej stronie linii biegnącej z wierzchołka go poprzedzającego do następnego, a gdy kąt jest większy od π to przeciwnie. Zarówno jak i w laboratorium 2 użyłem wyznacznika 3x3 własnej implementacji.

Poniżej znajdują się wykresy ze sklasyfikowanymi wybranymi figurami(dwie niemonotoniczne i dwie monotoniczne.

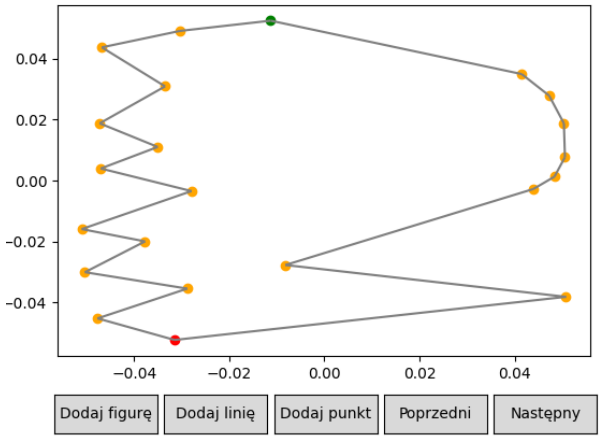
Wykres 4.1 Klasyfikacja zbioru D



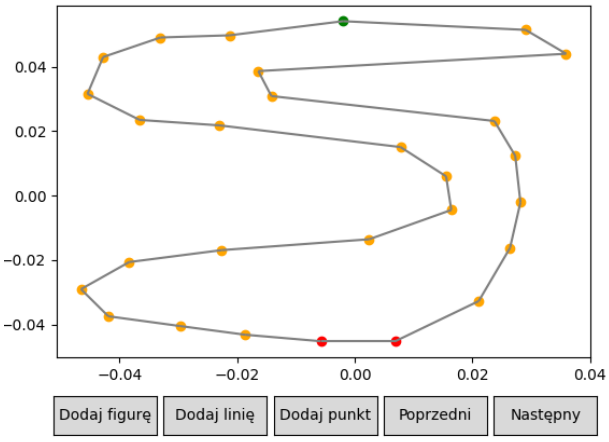
Wykres 4.2 Klasyfikacja zbioru E



Wykres 4.3 Klasyfikacja zbioru G



Wykres 4.4 Klasyfikacja zbioru H



5. Triangulacja wielokąta

Algorytm na początku przyjmuje listę punktów podaną w kolejności przeciwnej do ruchu wskazówek zegara. Po sprawdzeniu czy wielokąt jest monotoniczny, przy wykorzystaniu wcześniej użytej funkcji, można przystąpić do triangulacji. Do reprezentacji wielokąta wykorzystuje dwie listy *prev*, *after*, które dla każdego punktu przechowują informację o jego sąsiadach. Odpowiednio o wierzchołku go poprzedzającym i jego następniku w ruchu przeciwnym do wskazówek zegara.

Kolejnym krokiem jest znalezienie punktu o największej współrzędnej y, a także o najmniejszej współrzędnej y. Następnie muszę podzielić zbiór punktów na dwa podzbiory. W jednym będą wierzchołki, które znajdują się na ścieżce od wierzchołka najwyższego do najniższego, a drugie na ścieżce od wierzchołka najniższego do najwyższego. Dzięki kolejności w jakiej podane są punkty można łatwo dokonać tego podziału, po prostu przechodząc po tych ścieżkach. Jako że w głównej części algorytmu należy przechodzić po kolejnych punktach posortowanych po współrzędnej y. Oba uzyskane zbiory należy z powrotem scalić w jeden, lecz zachowując dla każdego punktu informacje do którego należy łańcucha. Dzięki temu, że wierzchołki w obrębie jednego łańcucha są już posortowane(wynika to z tego, że wielokąt jest y-monotoniczny) to do scalenia tych dwóch list można użyć algorytmu merge z merge-sorta.

Na tak przygotowanych punktach można zacząć główną część algorytmu. Na początku na stos wkładam dwa pierwsze wierzchołki. Następnie w głównej pętli przechodzę po kolejnych uporządkowanych po y wierzchołkach i dla każdego rozważam dwa przypadki:

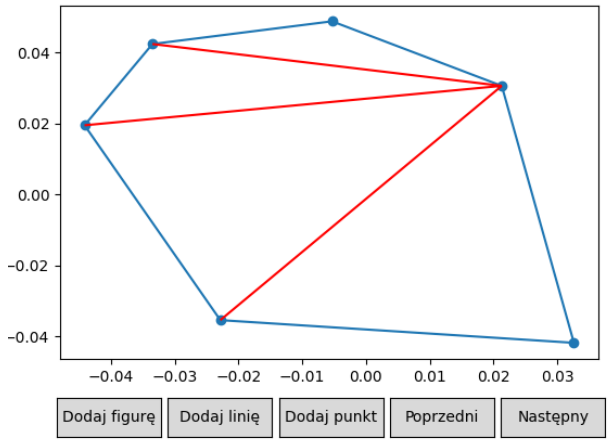
1. Kolejny wierzchołek leży na innym łańcuchy niż szczyt stosu – wtedy ten wierzchołek łączę ze wszystkimi innymi na stosie, jednocześnie zdejmując je ze stosu, na którym na końcu kładę dwa ostatnio zamiatane wierzchołki.
2. Kolejny wierzchołek jest na tym samym łańcuchu – wtedy analizuje kolejne trójkąty jakie nowy wierzchołek tworzy z dwoma znajdującymi się na stosie i jeśli trójkąt należy do wielokąta to dodaje krawędź do triangulacji i zdejmuje ostatni wierzchołek ze stosu. W przeciwnym przypadku na stosie kładę ostatni wierzchołek.

Uwagi końcowe do algorytmu:

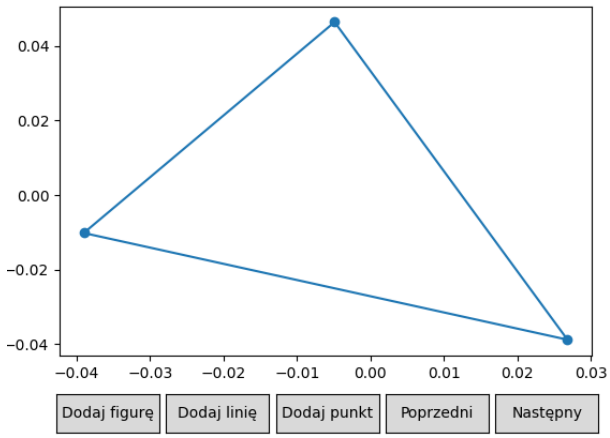
1. Jako, że wierzchołki początkowy i końcowy nie należały do żadnego z łańcuchów, należało dla nich rozpatrzyć oddzielne przypadki, gdyż mogło dojść do sytuacji, że algorytm do triangulacji dodawał krawędzie, które wcześniej należały do wielokąta.
2. Podczas stwierdzania, czy dany trójkąt jest w wielokącie użyłem wyznacznika. Musiałem rozpatrzyć tam przypadki ze względu na to czy nowy wierzchołek znajduje się w łańcuchy prawym czy lewym.
3. Triangulacje przechowywałem jako listę odcinków, gdyż podczas zamiatania krawędzie już rozpatrzone nie mogły wpływać w żaden sposób na wynik, więc użyłem najprostszej struktury.

Poniżej znajdują się wyniki triangulacji dla opisanych wcześniej zbiorów punktów.

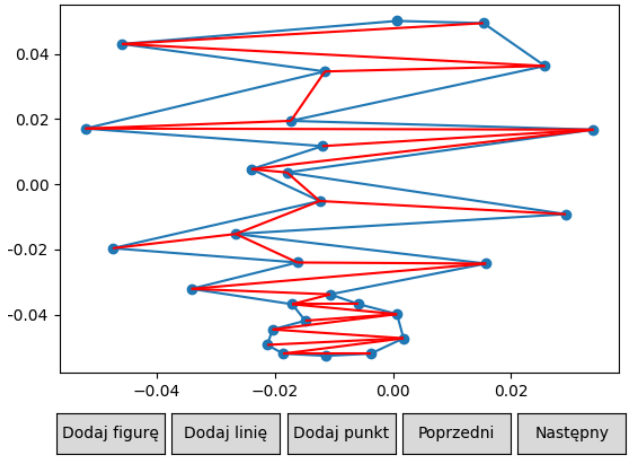
Wykres 5.1 Triangulacja zbioru A



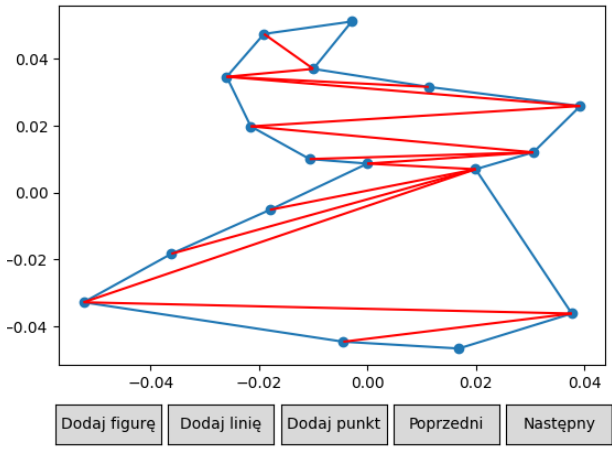
Wykres 5.2 Triangulacja zbioru B



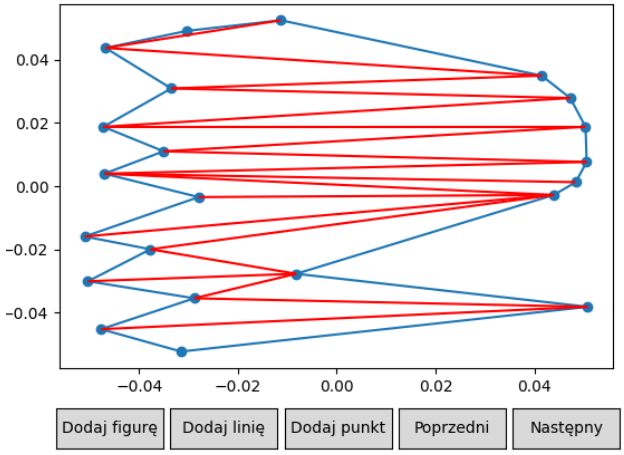
Wykres 5.3 Triangulacja zbioru C



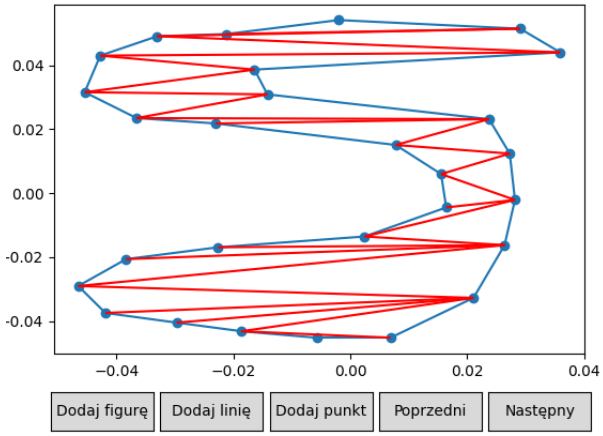
Wykres 5.4 Triangulacja zbioru F



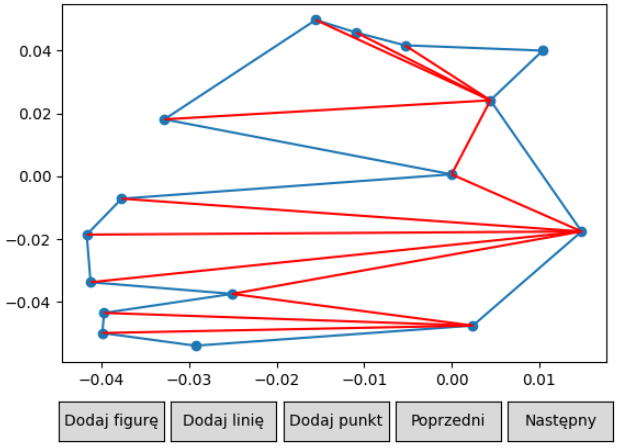
Wykres 5.5 Triangulacja zbioru G



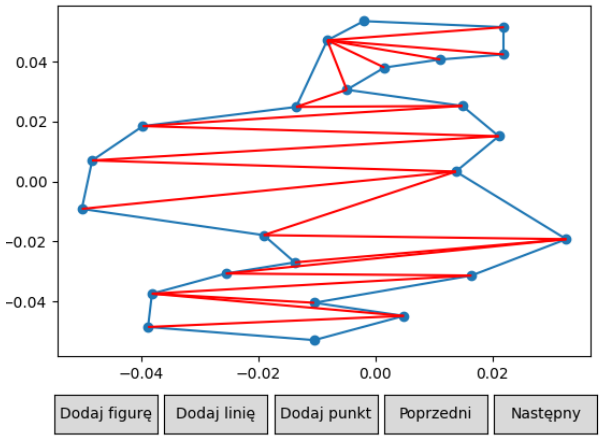
Wykres 5.6 Triangulacja zbioru H



Wykres 5.7 Triangulacja zbioru I



Wykres 5.8 Triangulacja zbioru J



Wizualizacja poszczególnych kroków algorytmu dostępna jest w pliku ipynb.

6. Wnioski

Algorytm zadziałał dla wszystkich zbiorów punktów, można zatem, że jest on poprawny. Ponadto, działa on w złożoności liniowej ze względu na ilość podanych punktów.