Algorytmy Geometryczne

Zadanie 3, laboratorium 3 - sprawozdanie

Radosław Kawa

1.Opis ćwiczenia

Na laboratorium 3 na początek należało dostosować aplikacje graficzna, aby można było zadawać wielokąty myszką oraz dodać ich odczyt i zapis. Następnie zaimplementować oraz przetestować działanie 3 algorytmów:

1.Procedura sprawdzająca czy podany wielokąt jest y-monotoniczny

2.Algorytm, który dla zadanego wielokąta będzie wyszukiwał wierzchołki początkowe, końcowe, łączące, dzielące i prawidłowe.

3.Procedura triangulacji wielokąta monotonicznego

2.Biblioteki oraz specyfikacja

Zadanie było wykonane w jupyter notebook, w języku Python. Wersja 3.10.0. Do rysowania wykresów zostało użyte narzędzie dostarczone na laboratorium, które wykorzystuje bibliotekę matplotlib. Umożliwiało to zadawanie wielokątów myszką.

Specyfikacja:

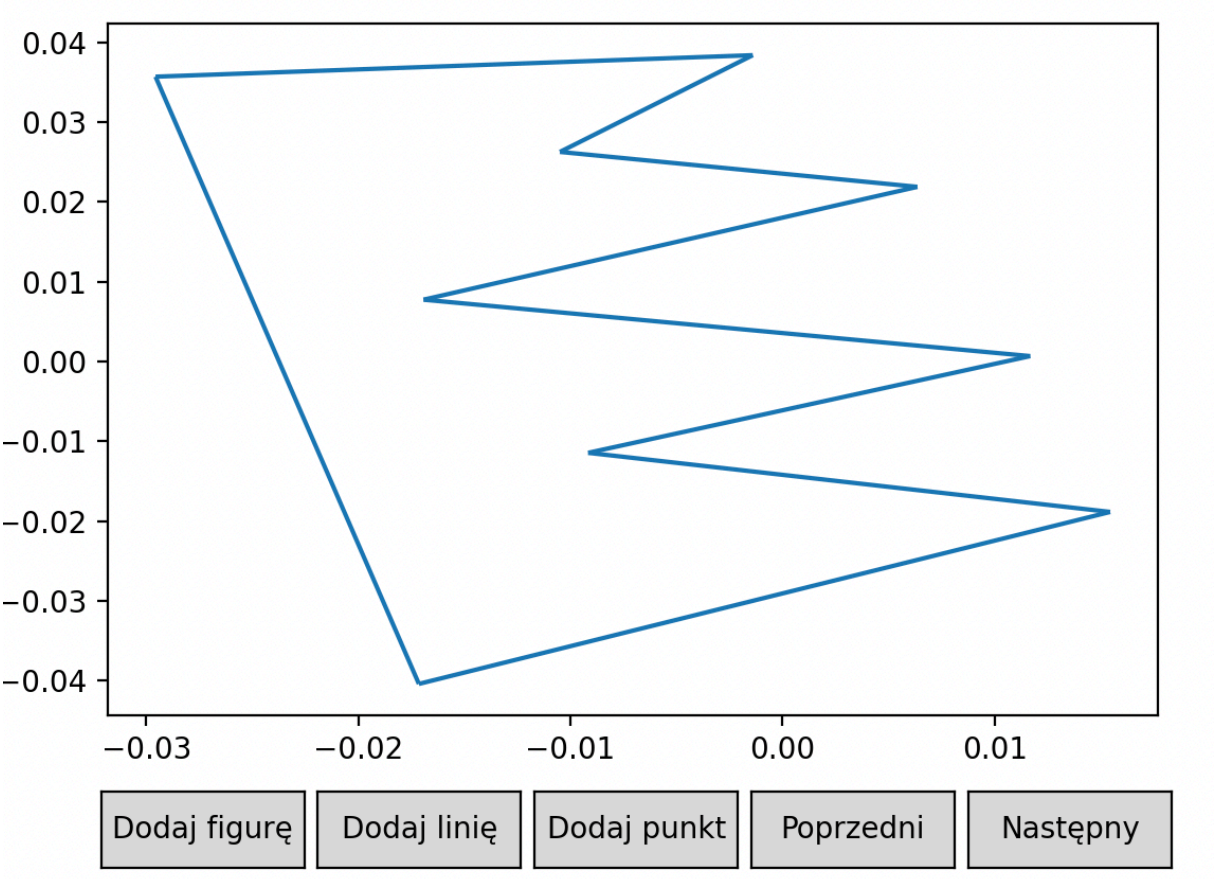
System operacyjny: macOS Montery
Procesor: Apple M1
Pamięć: 8GB

3. Opis algorytmów

a) Algorytm sprawdzający y-monotoniczność

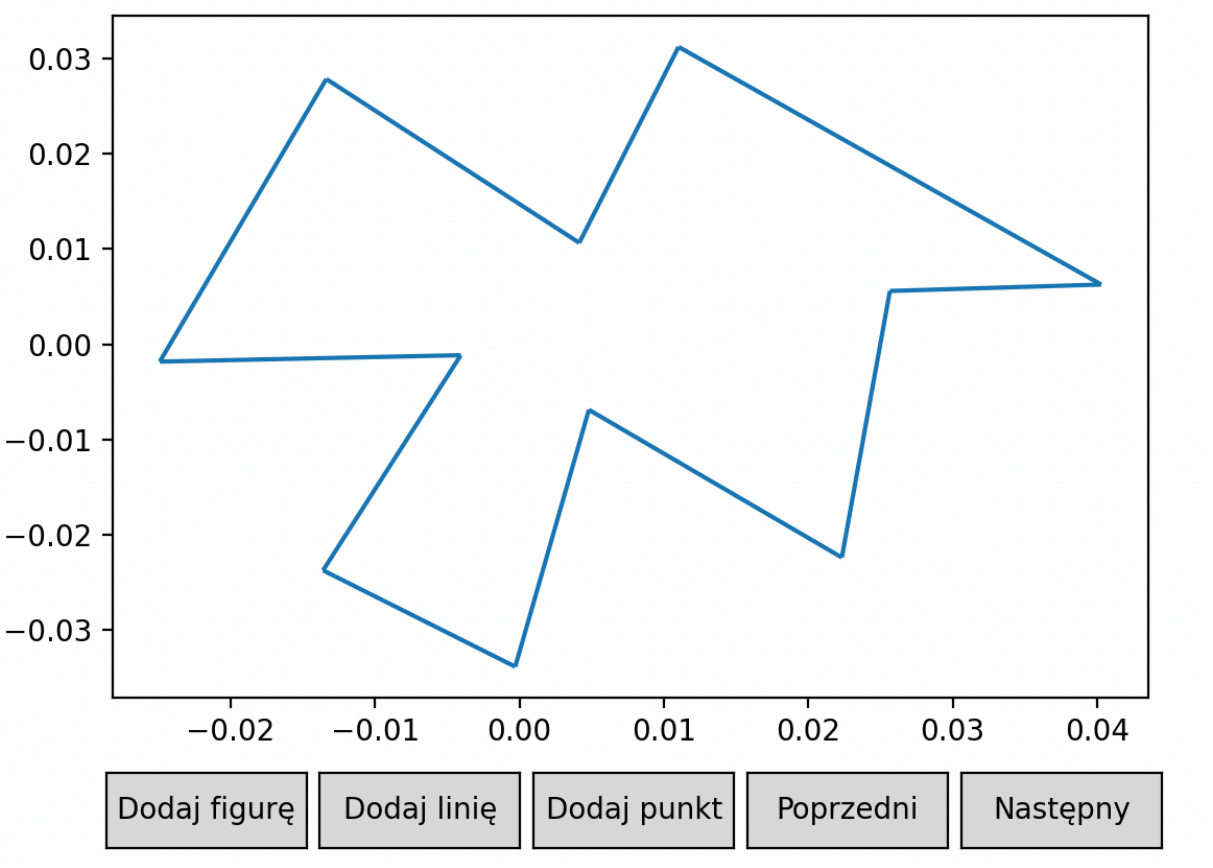
Pierwszym algorytmem do zaimplementowania był algorytm sprawdzający czy podany wielokąt jest y-monotoniczny. Polega on na przejściu po wszystkich wierzchołkach odwrotnie do wskazówek zegara tak jak był zadawany oraz sprawdzeniu czy każda 3 jest odpowiednia, to znaczy czy jej wyznacznik w tym przypadku jest dodatni (większy od zadanego epsilona)
Dodatkowo sprawdzam jeśli wyznacznik nie będzie dodatni to czy dana trójka jest poprawnie ułożona względem orientacji y. No i zwracała fałsz w zależności od tego czy dana trójka jest poprawnie ułożona. Ostatecznie cała funkcja zwracała fałsz gdy nie jest y-monotoniczny lub przeciwnie prawdę. Algorytm jest poprawny gdyż, wyznacznik nam gwarantuje poprawne ułożenie 3 punktów, a dodatkowo sprawdzenie poprawnego ułożenia 3 punktów gwarantuje nam to (w przypadku gdy wyznacznik będzie ujemny)

Przykładowo:



Wykres 1

Jest to wielokąt y-monotoniczny zakwalifikowany przez algorytm



Wykres 2

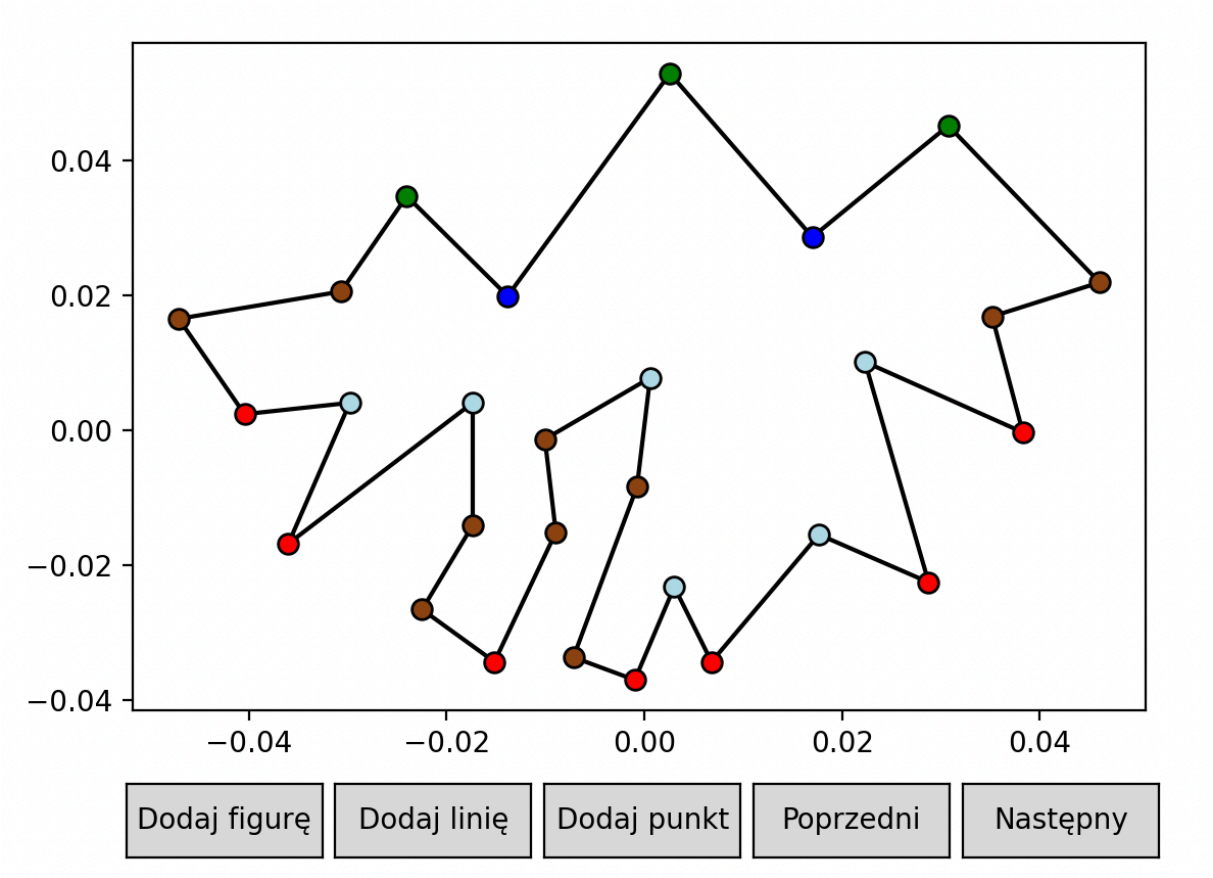
Natomiast ten wielokąt zostanie zakwalifikowany przez algorytm jako nie y-monotoniczny

b) Algorytm klasyfikujący wierzchołki

Algorytm przegląda wszystkie wierzchołki i określa na podstawie wysokości sąsiadów oraz kąta jaki z nimi tworzy ten wierzchołek:

- **początkowy**, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny < π,
- **końcowy**, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny < π,
- **łączący**, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny > π,
- **dzielący**, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny > π,
- **prawidłowy**, w pozostałych przypadkach (ma jednego sąsiada powyżej, drugiego – poniżej).

Przykładowo:



Wykres 3

Algorytm w tym wielokącie zakwalifikował wszystkie rodzaje wierzchołków

c) Algorytm wykonujący triangulacje

Na początku algorytm sprawdza czy wielokąt jest y-monotoniczny. Następnie algorytm dzieli te wierzchołki na lewy i prawy łańcuch, standardowo najniżej ułożony punkt ląduje do prawego łańcucha, natomiast najwyższy punkt do lewego. Następnie sortuje wierzchołki malejąco względem współrzędnej y. Pierwszy i drugi punkt lądują na stosie.
Na koniec przeglądamy wierzchołki z posortowanej listy oraz dodajemy z każdym krokiem krawędzie triangulacji pod warunkiem:

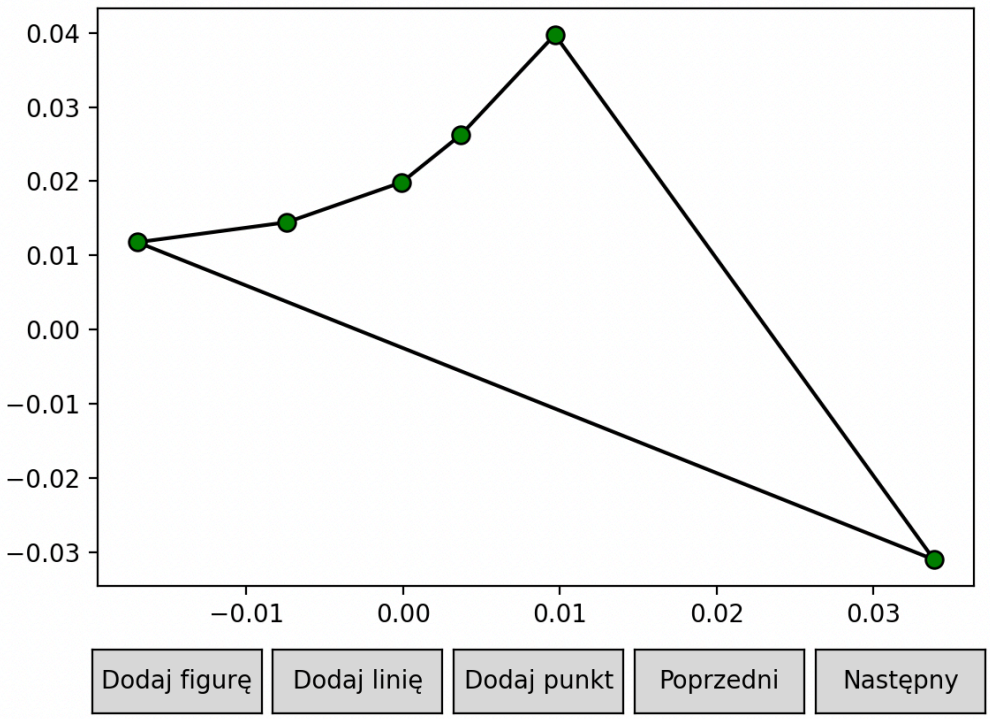
- Jeśli aktualnie rozpatrywany wierzchołek znajduje się w innym łańcuchu niż ten znajdujący się na szczycie stosu to łączymy go z wszystkimi wierzchołkami znajdującymi się aktualnie na stosie (pod warunkiem, że nie są to jego sąsiedzi), a na koniec na stosie zostawiam 2 ostatnie badane wierzchołki

- Jeśli aktualnie rozpatrywany wierzchołek znajduje się w tym samym łańcuchu co ten znajdujący się na szczycie stosu to:
    - Jeśli utworzony trójkąt nie należy do wielokąta to umieszczamy dwa wierzchołki na stosie (aktualny i ostatni badany ze stosu)
    - Jeśli utworzony trójkąt należy do wielokąta oraz szczyt stosu nie jest sąsiadem aktualnego wierzchołka to usuwamy ten wierzchołek ze stosu, a te wierzchołki łączymy krawędzią

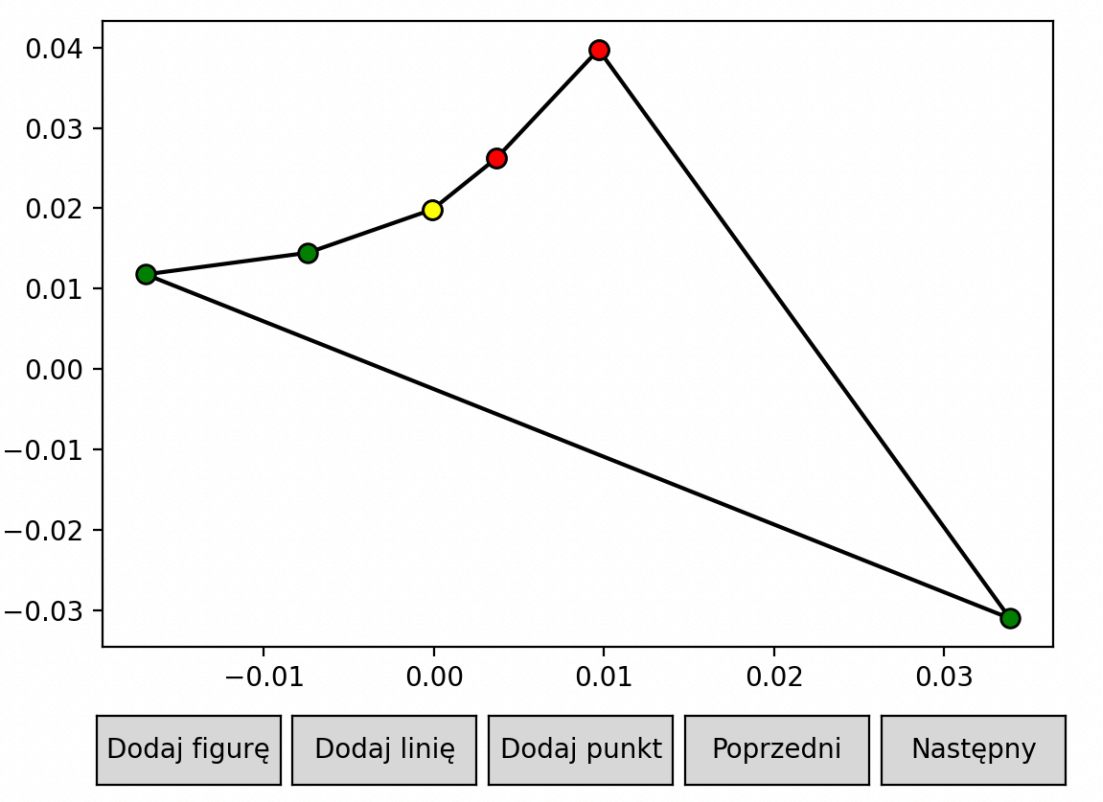
Algorytm ten może również zapisać przekątne do pliku przy podaniu odpowiedniego parametru, punkty są w zapisywane w następujący sposób, indeks danego punktu oraz punkt przykładowo: (0, (1.75,2.25)) – indeks 0
Natomiast przekątna przykładowo: [(0, (1.75,2.25)), (1, (2,3))]
Algorytm zwraca przekątne oraz sceny do wyświetlenia wykresu.
Dodatkowo koloruje wierzchołki w zależności od tego jakie zadanie miały w danym kroku, czerwony – stos, żółty – aktualny wierzchołek, fioletowy – połączony z wierzchołkiem żółtym, a zielony – wierzchołki nie brane pod uwagę w danym kroku. Niebieskim kolorem oznaczyłem przekątne.

Przykładowo:

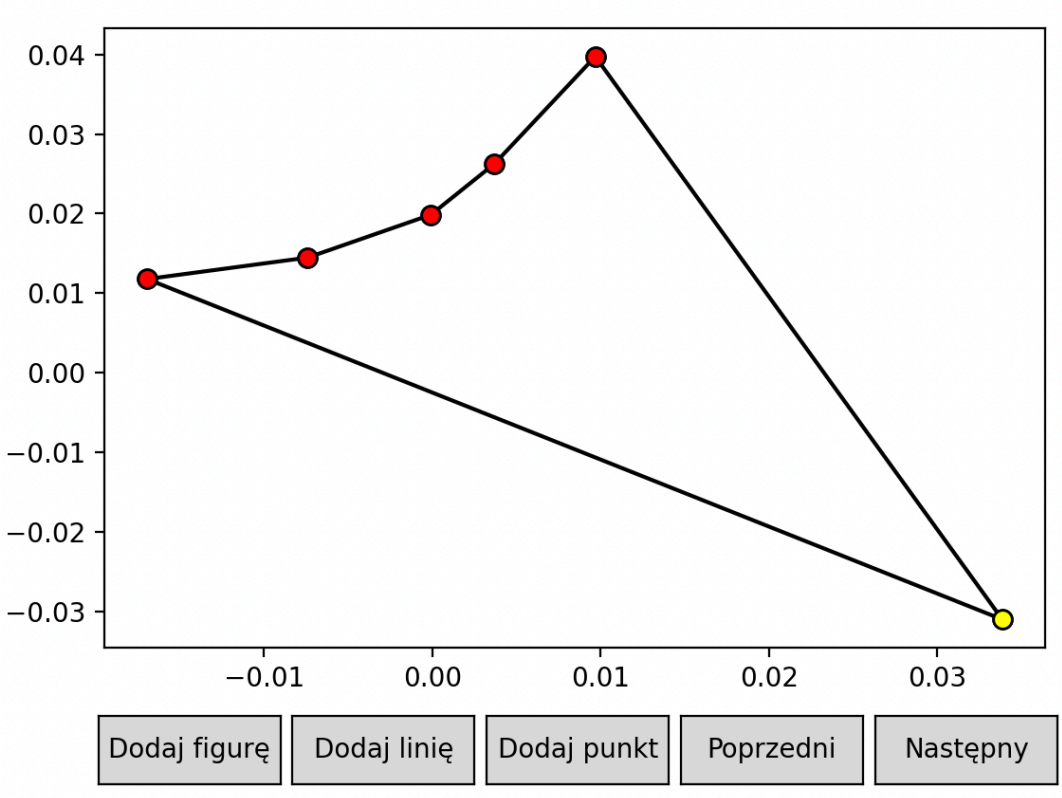
Wybrałem przykład z zajęć i przedstawiłem kroki w algorytmie (nie są to wszystkie kroki następują po kolei, natomiast pomijałem mniej istotne)



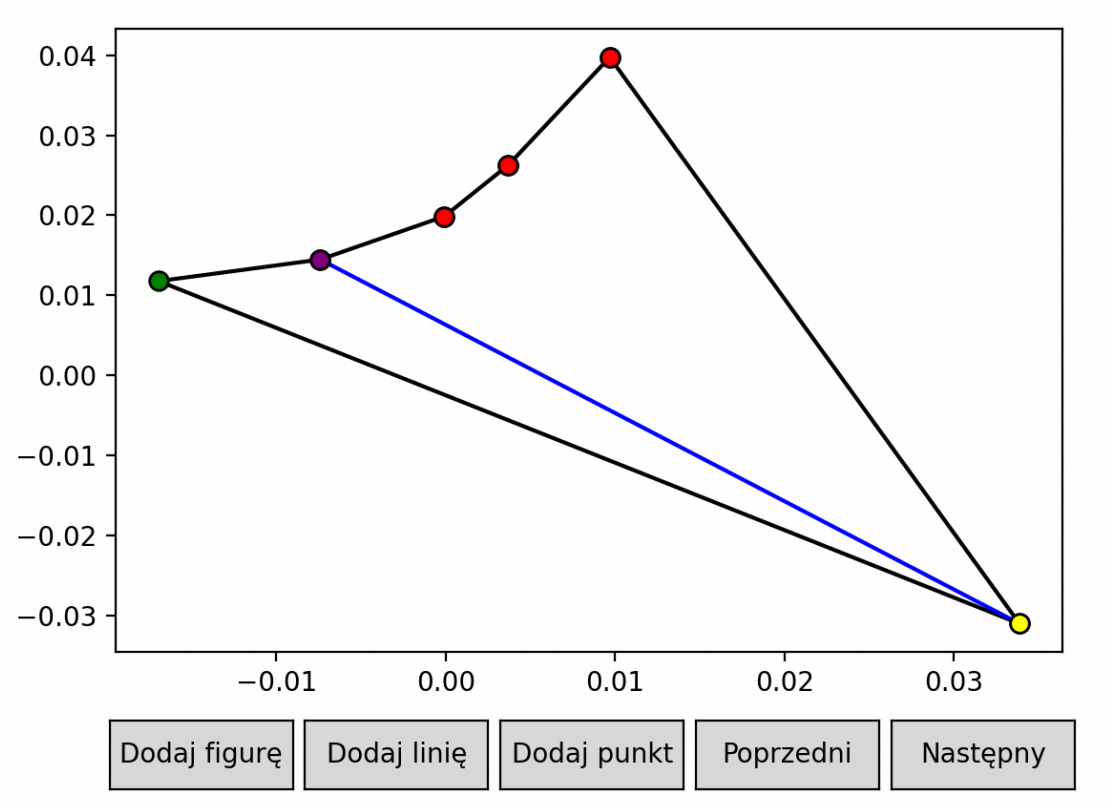
Wykres 4



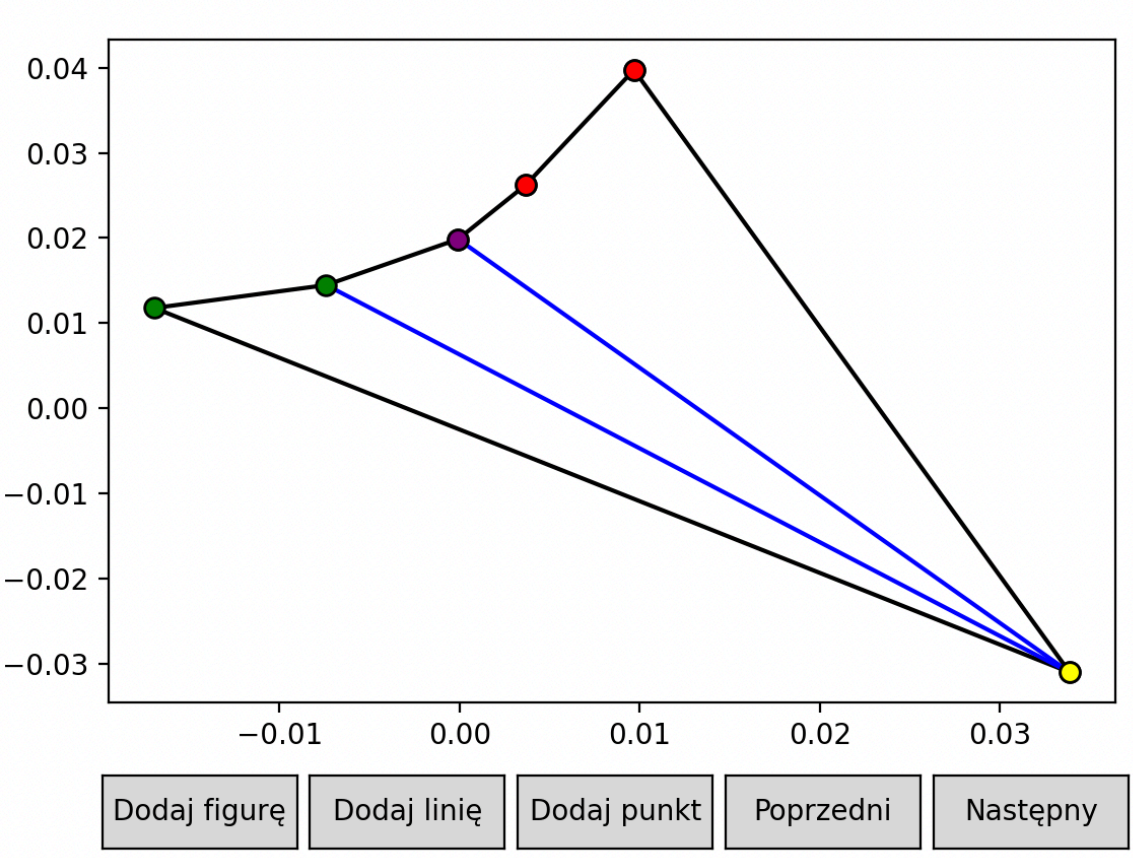
Wykres 5



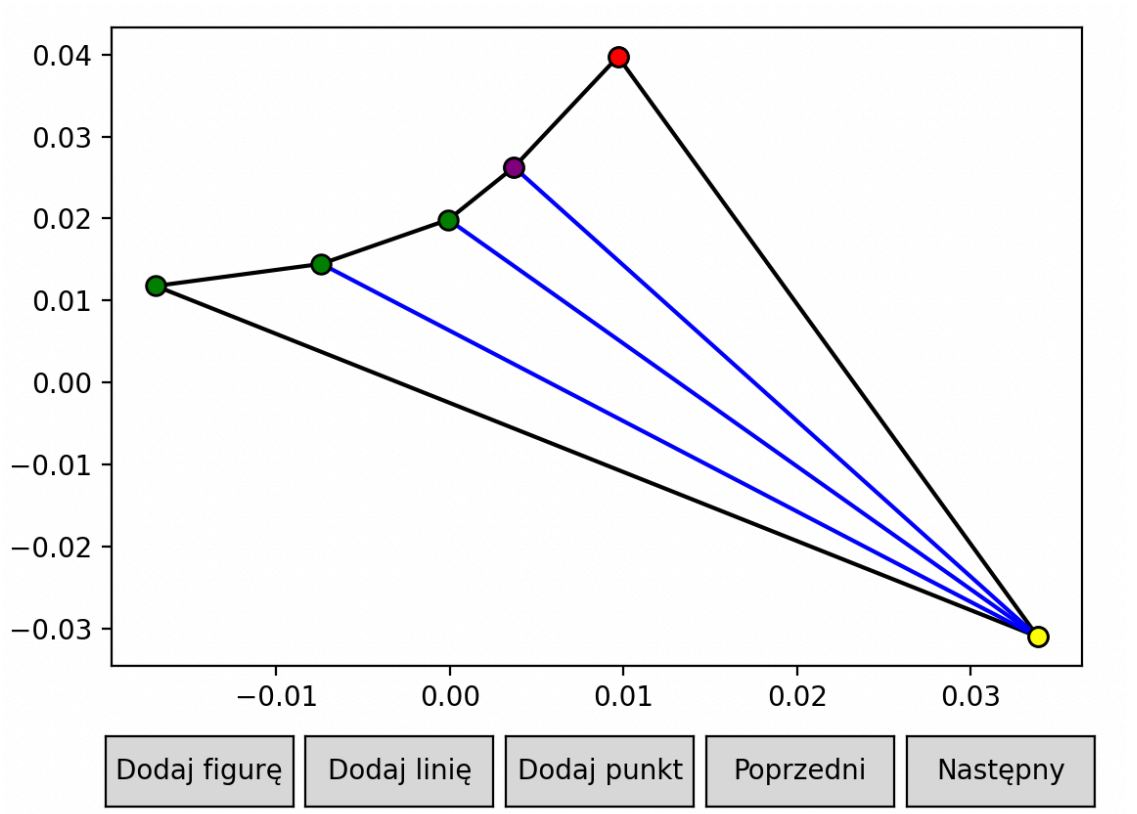
Wykres 6



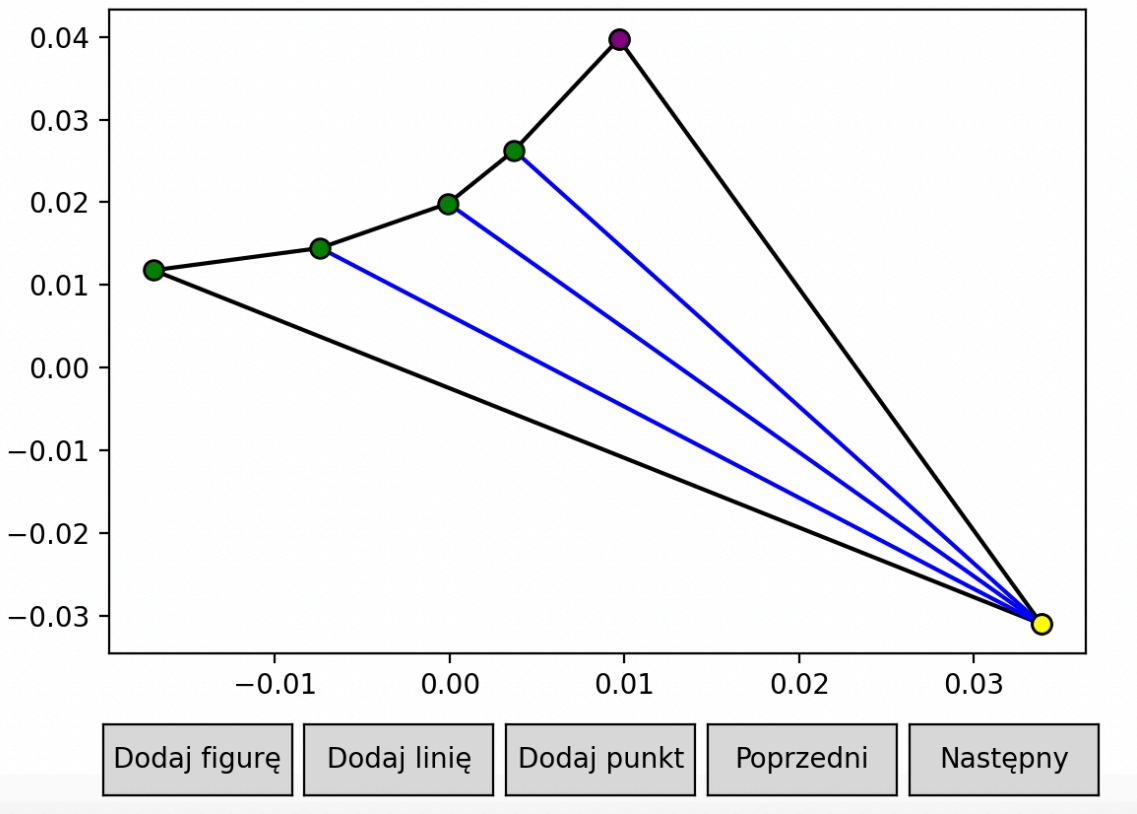
Wykres 7



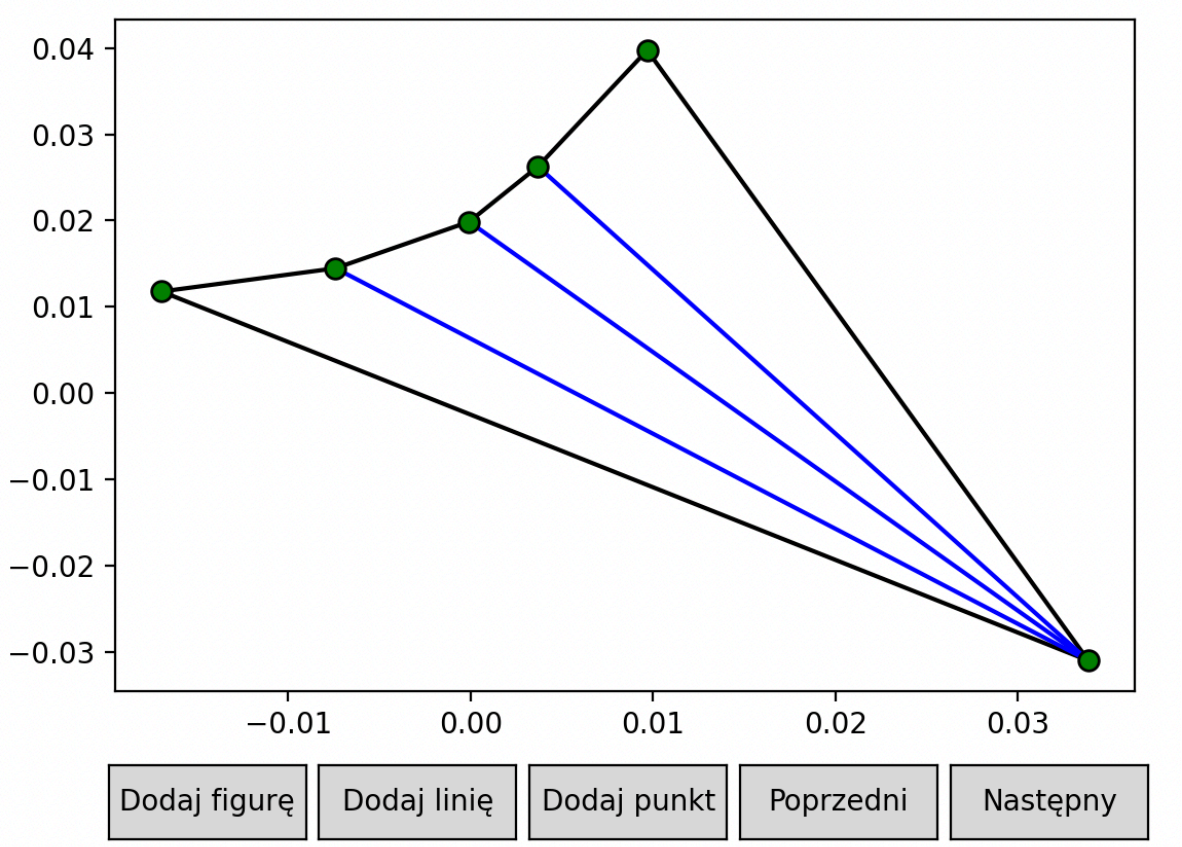
Wykres 8



Wykres 9

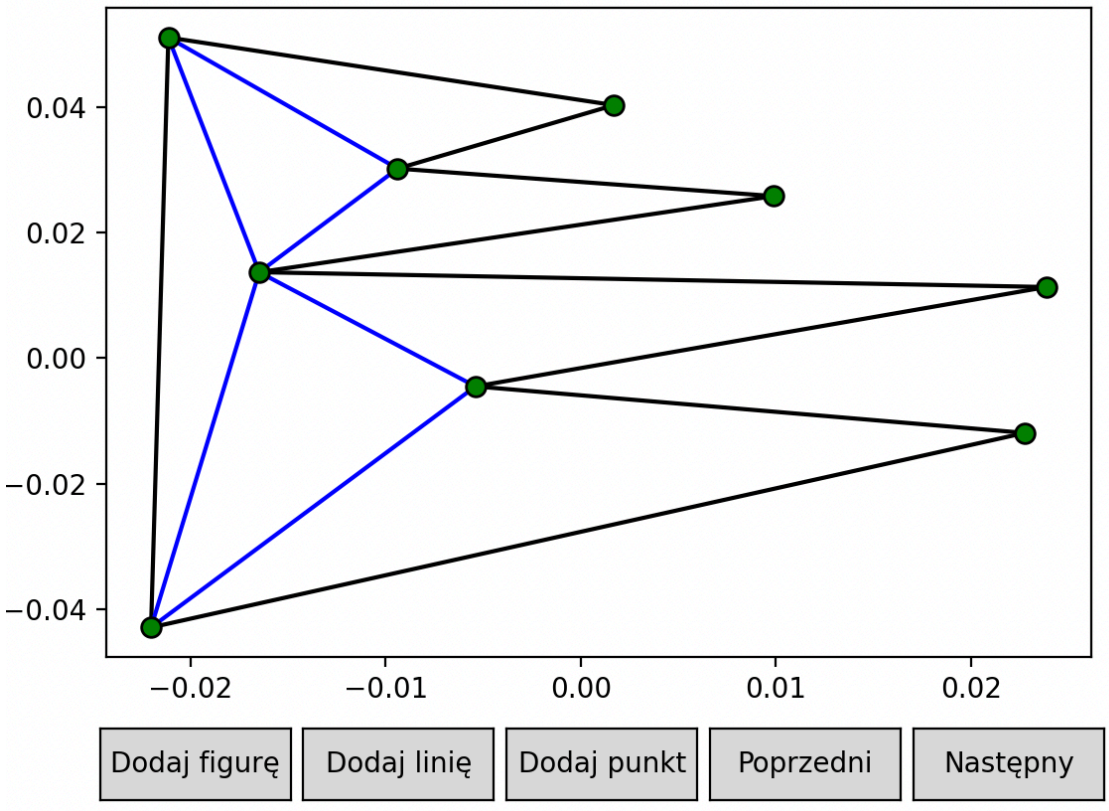


Wykres 10

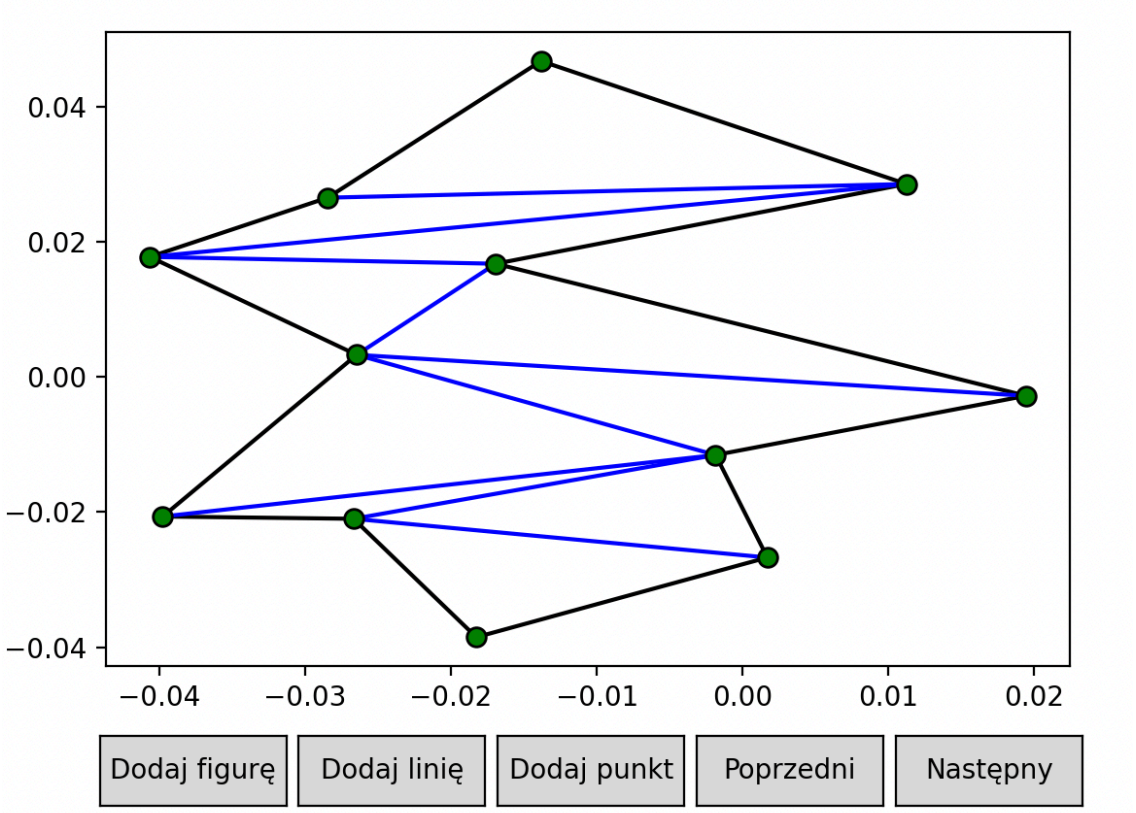


Wykres 11

Przedstawiłem dodatkowo wyniki końcowe wybranych wielokątów:



Wykres 12



Wykres 13

Wybierałem wykresy w zależności od tego jak algorytm się zachowuje, więc w szczególności dodałem wykresy z zajęć, gdyż one zawierają przypadki, które mogą kwalifikować przekątne, które nie powinny być kwalifikowane w przypadku złego algorytmu (będą się wtedy przecinać w danym miejscu przekątne). Głównie problem by się nadarzył, gdyby algorytm nie posiadał sprawdzania sąsiadów, bo wtedy mogłyby być przekątne na obwodach wielokąta. Dodatkowo, niektóre sprawdzają czy odpowiednio wierzchołki są ściągane ze stosu (przypadek, gdy mogą być źle ściągane np. nie w pętli, to by zaistniało przecięcie danych przekątnych co nie powinno mieć miejsca)

4. Wnioski

Algorytmy działają poprawnie na powyższych zbiorach, nie zauważyłem żadnego błędu w implementacji bazując na tych wykresach. Można stwierdzić, że zostały zaimplementowane poprawnie.