Juliusz Wasieleski
Informatyka II rok, grupa 1

Algorytmy geometryczne, laboratorium 3 sprawozdanie

1.Opis ćwiczenia

W zadaniu numer 3 należało zaimplementować algorytm sprawdzania, czy dany wielokąt jest y-monotoniczny. Po tym powinna nastąpić implementacja algorytmu klasyfikującego wierzchołki w danym wielokącie, a na samym końcu implementacja algorytmu odpowiedzialnego za triangulację wielokąta y-monotonicznego na podstawie algorytmu z wykładu. Należało również dostosować aplikację graficzną w taki sposób, aby możliwe było zadawanie wielokątów przy użyciu myszki, ich zapis oraz odczyt.

2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonałem w Jupyter Notebook i napisałem w języku Python. Podczas wykonywania zadania korzystałem z bibliotek *pandas* oraz *numpy.*

Do rysowania wykresów użyłem narzędzia graficznego z laboratorium, które jest oparte o bibliotekę *matplotlib*.

Podczas kolorowania wierzchołków przyjąłem konwencję z wykładu:

- **Zielony** – wierzchołek początkowy (starting),
- **Czerwony** – wierzchołek końcowy (ending),
- **Granatowy** – wierzchołek łączący (connective),
- **Błękitny** – wierzchołek dzielący (separative),
- **Brązowy** – wierzchołek prawidłowy (correct).

Wszystkie obliczenia prowadziłem na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

3. Plan i sposób wykonania ćwiczenia

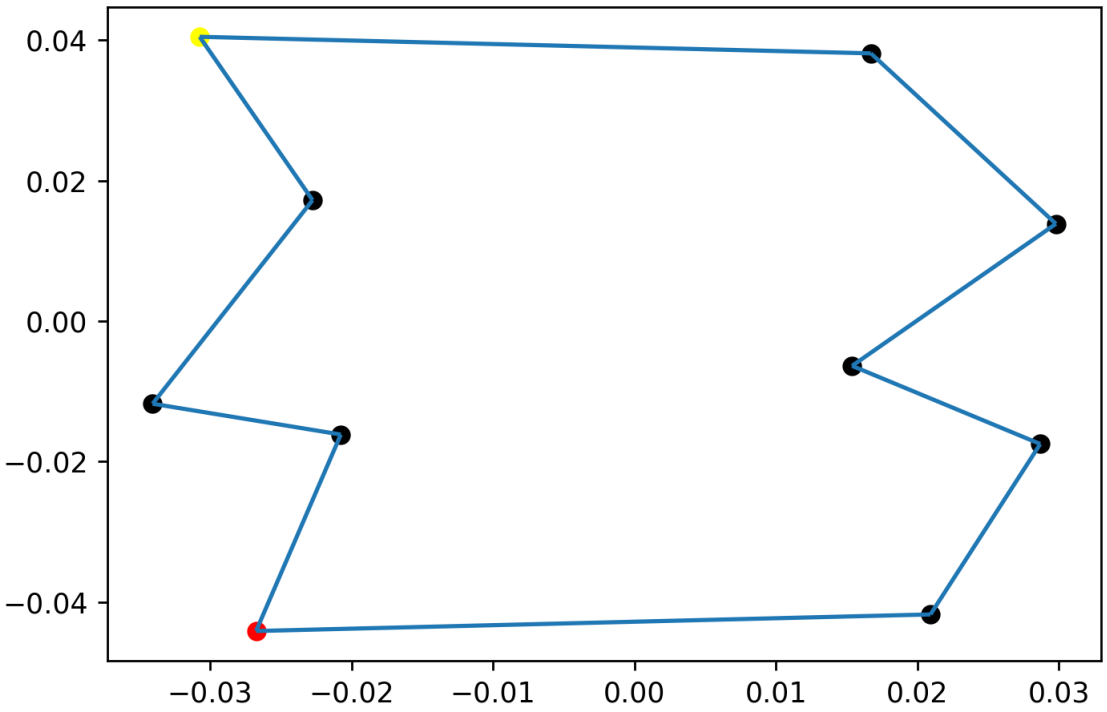
1. Przygotowuję funkcje do wczytywania punktów zaznaczonych myszką, zapisu ich do pliku i wczytywania punktów z pliku
2. Implementuję algorytm sprawdzający y-monotoniczność wielokąta i sprawdzam, czy jest on poprawny
3. Implementuję algorytm klasyfikujący wierzchołki wielokąta i sprawdzam jego poprawność
4. Implementuję algorytm wykonujący triangulację y-monotonicznego wielokąta

4. Wykonanie ćwiczenia

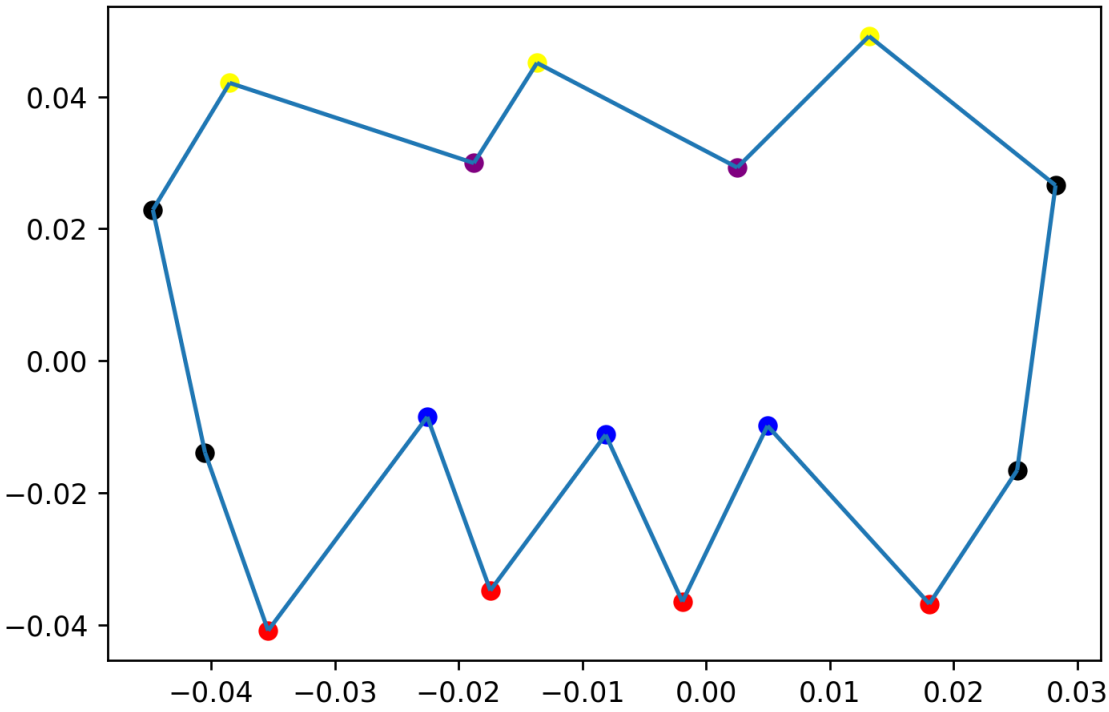
4.1 Algorytm sprawdzający y-monotoniczność

W moim rozwiązaniu powyższe zagadnienie realizuje funkcja *y_monotonic.* Ten algorytm polega na przejściu po wszystkich wierzchołkach wielokąta przeciwnie do ruchu wskazówek zegara i sprawdzeniu, czy każda trójka kolejnych punktów (aktualnie sprawdzany punkt, jego wcześniejszy i kolejny sąsiad) nie tworzą wierzchołka łączącego lub dzielącego. Funkcja *is_connective_or_separative* wykonuje to sprawdzenie. Jeżeli którakolwiek trójka takowy tworzyła, to znaczy, że zadany wielokąt nie jest wielokątem y-monotonicznym, więc jego główna funkcja zwraca fałsz. Natomiast jeżeli algorytm przejdzie przez punkty i nie napotka żadnego punktu wierzchołka łączącego lub dzielącego to znaczy, że jest y-monotoniczny, a więc funkcja zwraca prawdę.

Przykład działania algorytmu:



(rys. 1) Wielokąt, dla którego algorytm zwrócił prawdę



(rys.2) Wielokąt, dla którego algorytm zwrócił fałsz

Powyższe wykresy zostały zadane przy użyciu narzędzia graficznego w celu przetestowania poprawności funkcji sprawdzającej y- monotoniczność, a następnie zapisane do plików testowych. Rysowanie wykonała funkcja, która od razu zaklasyfikowała wierzchołki.

4.2 Algorytm klasyfikujący wierzchołki wielokąta

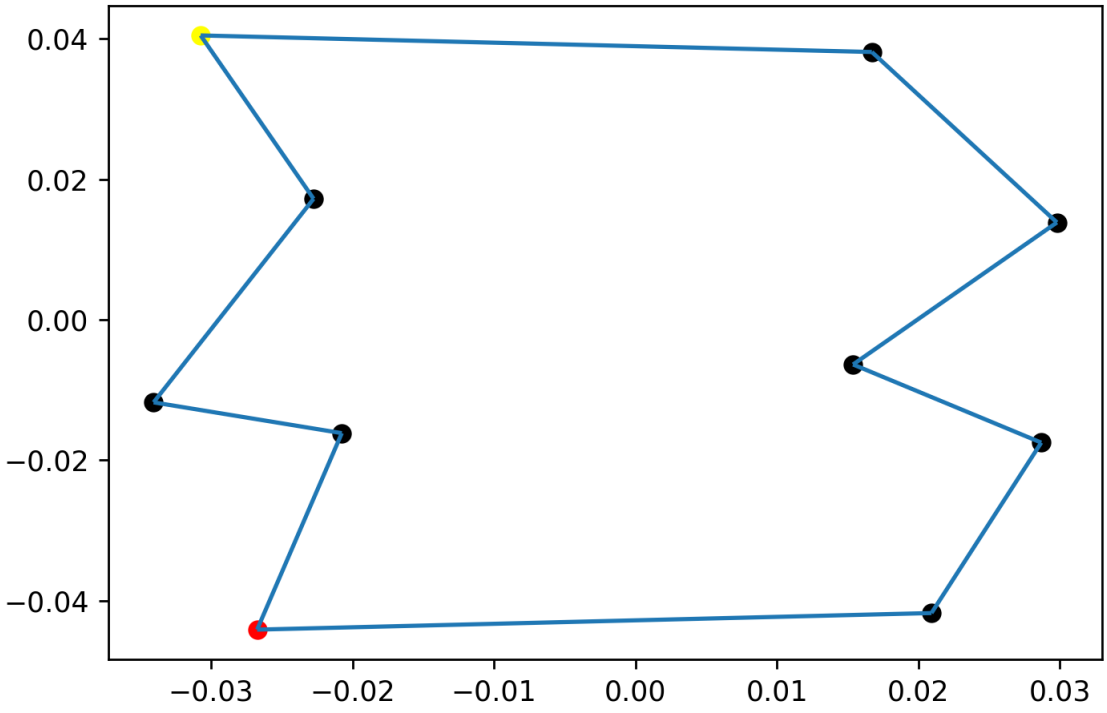
Algorytm polega na przejściu po wszystkich wierzchołkach wielokąta i określeniu na podstawie wysokości sąsiadów aktualnie sprawdzanego wierzchołka oraz kąta wewnętrznego wielokąta jaki tworzy ten wierzchołek. W mojej implementacji wykorzystałem do tego własnoręcznie napisaną funkcję do

obliczenia wyznacznika macierzy 3x3, dzięki czemu mogę sprawdzić wzajemne położenie trzech punktów. Następnie sprawdzam, które warunki są spełnione, żeby dobrze sklasyfikować wierzchołek:

- Wierzchołek początkowy – obaj sąsiedzi leżą poniżej, kąt wewnętrzny tworzony przez wierzchołek i jego sąsiadów jest < π,
- Wierzchołek końcowy – obaj sąsiedzi leżą powyżej, kąt wewnętrzny tworzony przez wierzchołek i jego sąsiadów jest < π,
- Wierzchołek łączący – obaj sąsiedzi leżą powyżej, kąt wewnętrzny tworzony przez wierzchołek i jego sąsiadów jest > π,
- Wierzchołek dzielący – obaj sąsiedzi leżą poniżej, kąt wewnętrzny tworzony przez wierzchołek i jego sąsiadów jest > π,
- Wierzchołek prawidłowy – w pozostałych przypadkach.

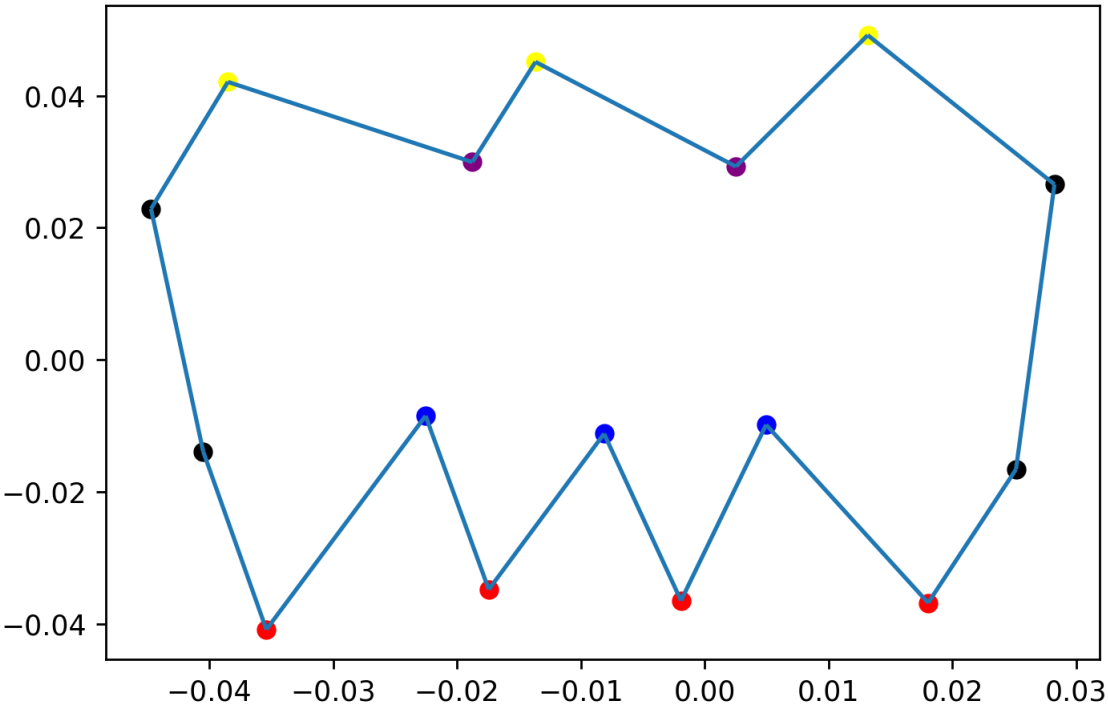
Przykłady działania algorytmu:

```
Poczatkowych: 1
Koncowych: 1
Lacznie wierzcholkow: 10
Poczatkowych: 1
Koncowych: 1
Łączących: 0
Dzielących: 0
Prawidłowych: 8
Lacznie wierzcholkow: 10
```



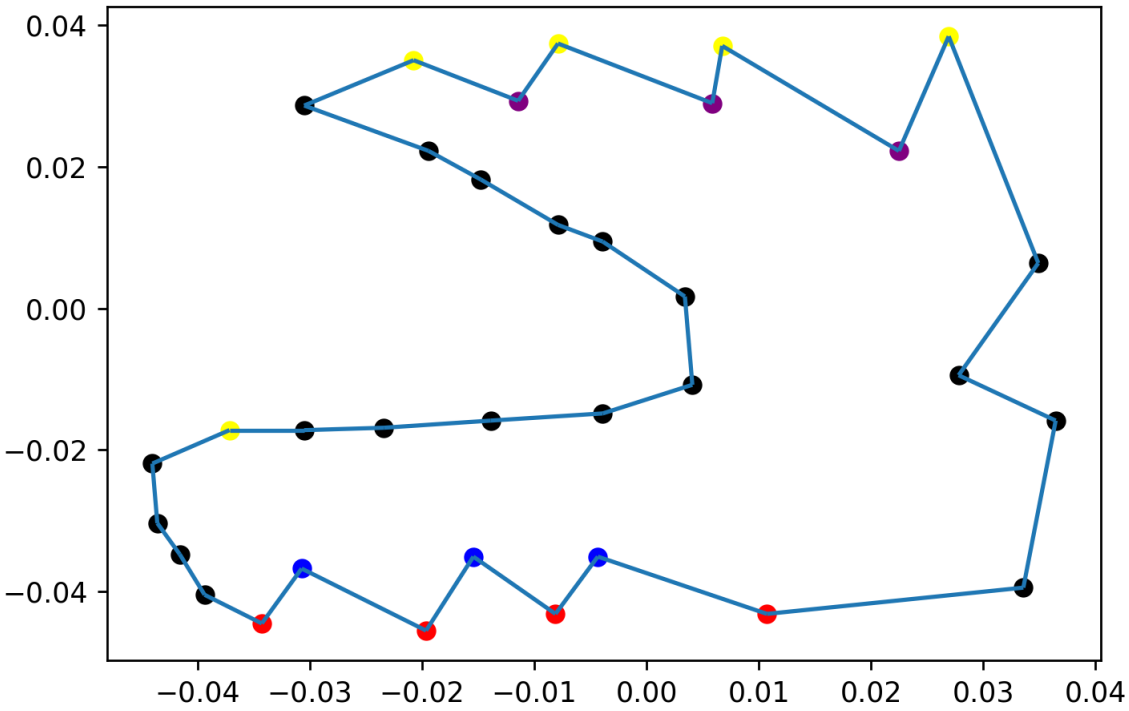
(rys. 3) Wynik działania algorytmu dla testu nr 7

```
Poczatkowych: 3
Koncowych: 4
Lacznie wierzcholkow: 16
Poczatkowych: 3
Koncowych: 4
Łączących: 2
Dzielących: 3
Prawidłowych: 4
Lacznie wierzcholkow: 16
```



(rys.4) Wynik algorytmu dla testu nr 8

```
Poczatkowych: 5
Koncowych: 4
Lacznie wierzcholkow: 34
Poczatkowych: 5
Koncowych: 4
Łączących: 3
Dzielących: 3
Prawidłowych: 19
Lacznie wierzcholkow: 34
```



(rys.5) Wynik działania algorytmu dla testu 9

Powyższe wykresy zostały zadane przy użyciu narzędzia graficznego w celu przetestowania poprawności funkcji klasyfikującej wierzchołki, a następnie zapisane do plików testowych.

4.3 Algorytm wykonujący triangulację

Algorytm na samym początku sprawdza, czy wielokąt jest y-monotoniczny, jeżeli nie jest, to kończy pracę z komunikatem „Wielokąt nie jest monotoniczny!”. Sprawdzanie odbywa się przy pomocy funkcji *y_monotonic*. Następnie za pomocą funkcji *map_to_points_and_lines* punkty zamieniam na elementy klasy *Point.* Klasa ta, ma pola opowiadające za informacje o tym, z którego łańcucha jest dany punkt oraz jaki jest jego indeks w danych wejściowych. Po zmianie punktów, uzupełniam ich pola, co jest równoznaczne ze stworzeniem dwóch łańcuchów. Najwyższy wierzchołek trafia do łańcucha lewego, czyli zstępującego, a najniższy do prawego czyli wstępującego. Po posortowaniu pierwszy oraz drugi element listy wrzucam na stos. Przeglądam wszystkie pozostałe wierzchołki z posortowanej listy, a następnie w zależności od przypadku wybieram wariant działania algorytmu.

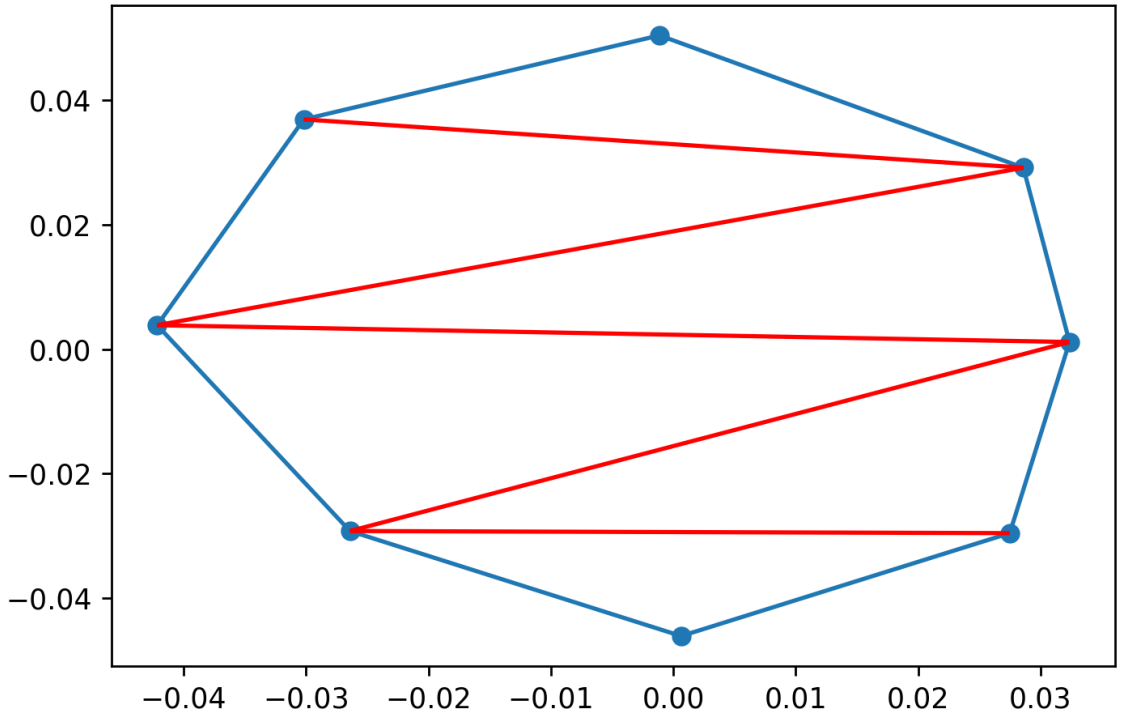
- **Wariant 1 – wierzchołek należy do innego łańcucha niż wierzchołek z góry stosu**
  W takiej sytuacji można utworzyć przekątne ze wszystkimi wierzchołkami znajdującymi się na stosie. Zdejmuję więc wszystkie, następnie po zakończeniu analizy na stos odkładane są aktualnie analizowany wierzchołek oraz wierzchołek, który leżał na samym szczycie stosu przed rozpoczęciem operacji usuwania wierzchołków ze stosu.

• **Wariant 2 – wierzchołek należy do tego samego łańcucha co wierzchołek z góry stosu**
W takiej sytuacji ściągam ze stosu dwa wierzchołki i analizuję czy trójkąt złożony z nich w pełni należy do wielokąta. Jeżeli tak, to tak długo jak taki stan się utrzymuje, ściągam ze stosu kolejne wierzchołki i je analizuję w taki sam sposób. W wypadku, w którym trójkąt nie należy do wielokąta odkładam wszystkie 3 analizowane wierzchołki na stos (dwa ze stosu i jeden aktualnie analizowany). Jeżeli dojdę do momentu w którym stos jest pusty lub któryś kolejny analizowany trójkąt nie należy do wielokąta, to odkładam na stos ostatnio ściągnięty wierzchołek oraz aktualnie analizowany wierzchołek.

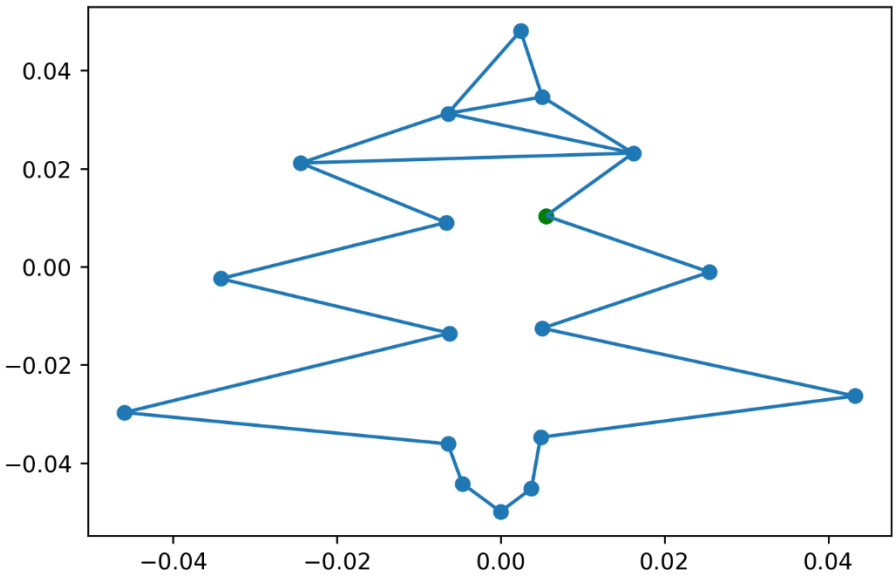
Aby uniknąć dodatkowych przekątnych, sprawdzam czy dwa wierzchołki, które aktualnie badam, nie są swoimi sąsiadami. Odpowiedzialna jest za to funkcja *do_not_follow*, w której sprawdzam, czy wierzchołki, między którymi chcę dodać przekątną nie mają kolejnych indeksów. Indeksy te, są zapisane w polu index dla każdego punktu. Sprawdzam też wtedy, czy nie mam do czynienia z pierwszym i ostatnim punktem, ponieważ różnica ich indeksów nie jest równa 1, tylko n-1, gdzie n to ilość punktów, a i tak już mamy krawędź między nimi.

Funkcja dokonująca triangulacji zwraca sceny (potrzebne do wizualizacji triangulacji) oraz listę przekątnych. Przekątne są na niej reprezentowane jako krotki dwuelementowe, które na obu polach mają liczby całkowite będące indeksami punktów, między którymi prowadzę daną przekątną.

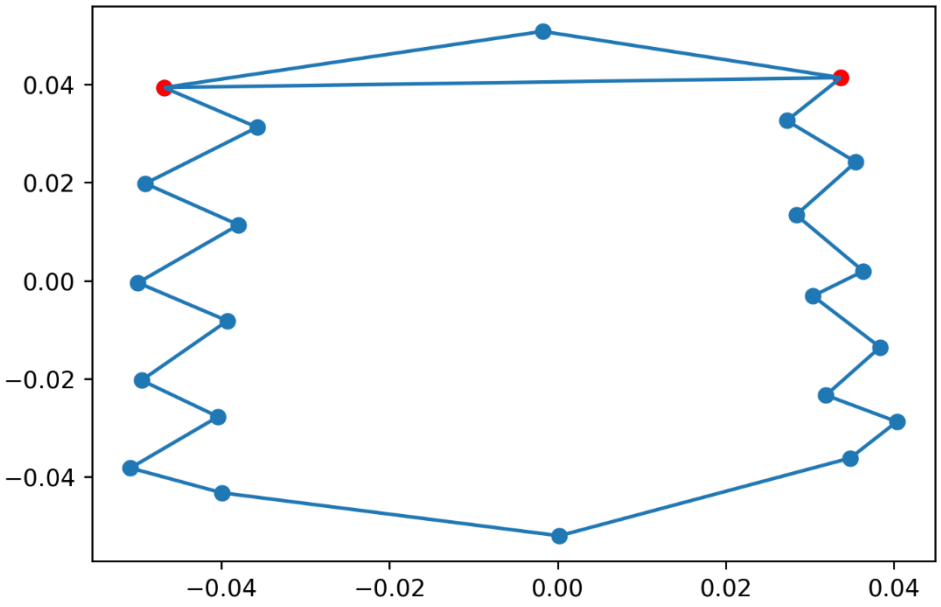
Przykłady działania algorytmu (niektóre kroki):



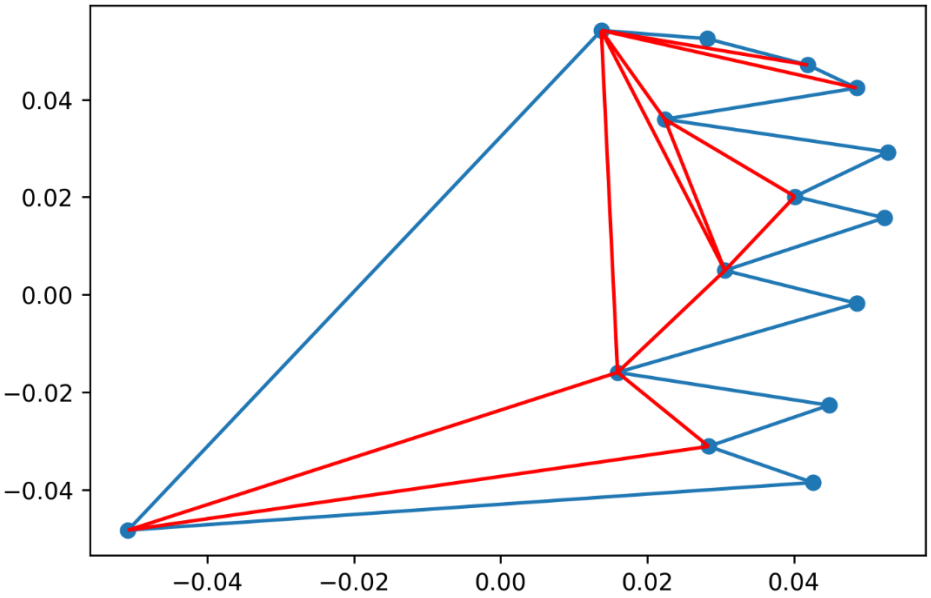
(rys.6) Zakończona triangulacja dla testu nr 1



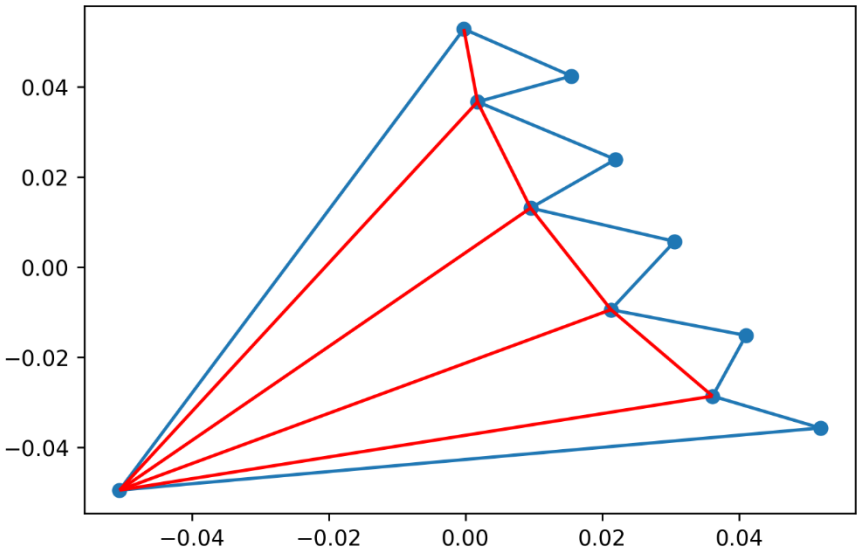
(rys. 7) Sprawdzanie kolejnych punktów



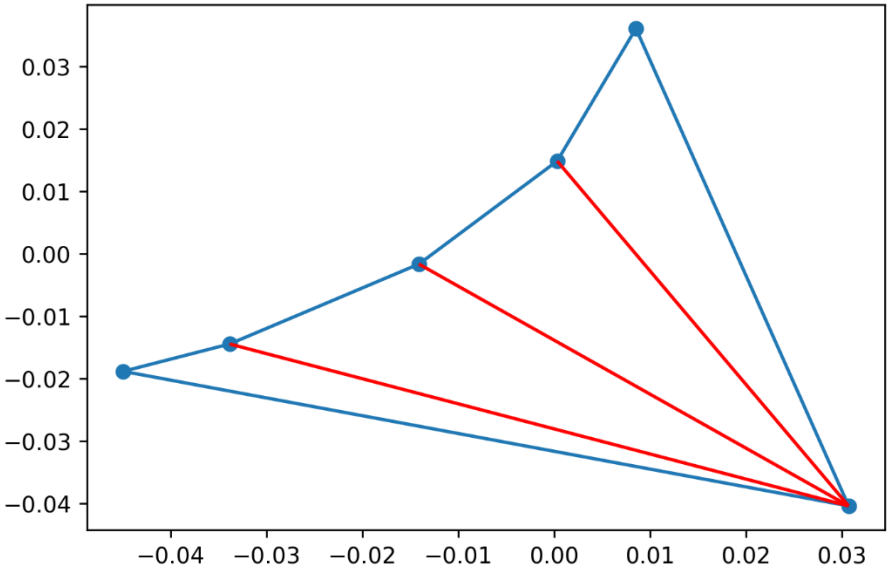
(rys. 8) Moment dodawania kolejnej przekątnej



(rys. 9) Zakończona triangulacja dla testu nr 4



(rys. 10) Zakończona triangulacja dla testu nr 5



(rys. 11) Zakończona triangulacja dla testu nr 10

6. Wnioski

Algorytmy po ich przetestowaniu na powyższych zbiorach działają poprawnie. Biorąc pod uwagę to, że w wielu wypadkach są to przypadki skrajne, można stwierdzić, że algorytmy są zaimplementowane poprawnie i działają zgodnie z oczekiwaniami.