Algorytmy Geometryczne
Sprawozdanie 3

Aga Patro
czw_13.30_A

1. Specyfikacja sprzętu i narzędzia wykorzystane w realizacji

System: Debian Linux Parrot OS x64
Procesor: AMD Ryzen 5 4500U, 6 rdzeni, 6 wątków, 4.00GHz
Pamięć RAM: 16 GB
Środowisko: Jupyter Notebook
Język: Python 3
Narzędzie pomocnicze: plik *geometria.ipynb* dostarczony z poleceniem, biblioteki *numPy*, *math, random*, *matplotlib* oraz *time*

2. Temat ćwiczenia

Celem ćwiczenia było zapoznanie się z tematem monotoniczności wielokątów oraz zaimplementowanie trzech algorytmów: algorytmu który klasyfikuje wierzchołki wielokątów, algorytmu sprawdzającego monotoniczność wielokąta względem osi OY, oraz algorytmu wykonującego triangulację wielokątów y-monotonicznych.

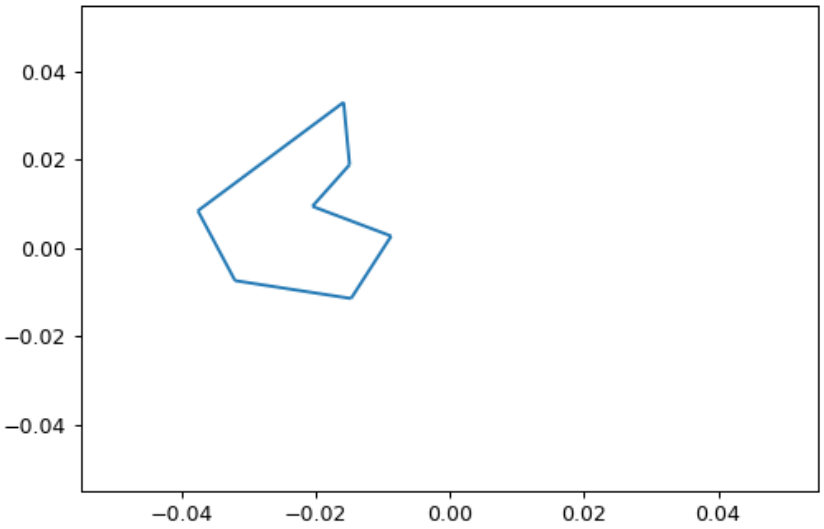
3. Realizacja

3.1. Algorytm sprawdzający czy wielokąt jest y-monotoniczny

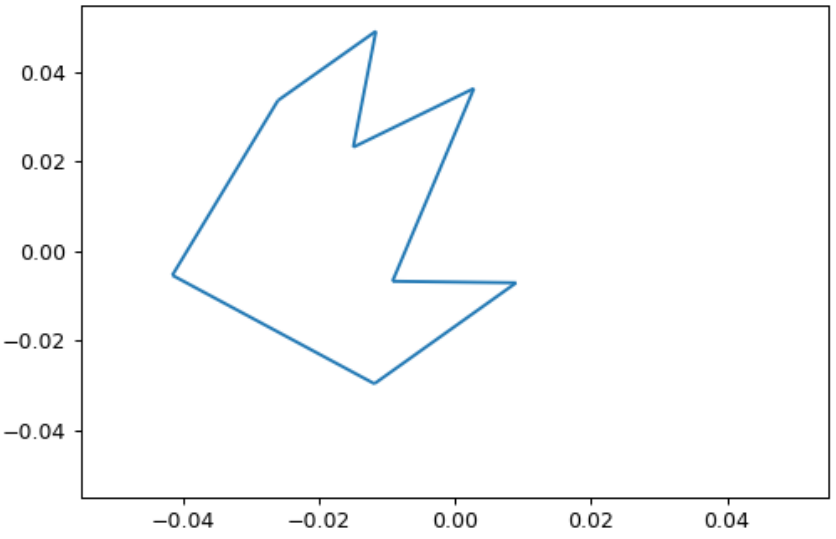
Wielokątem monotonicznym nazywamy taki wielokąt, który nie ma wierzchołków podziału ani łączenia. Wierzchołkiem podziału nazywamy wierzchołek, dla którego obaj sąsiedzi leżą poniżej niego i wewnętrzny kąt w wierzchołku v jest większy od $\pi\,(180^\circ)$. Wierzchołkiem łączenia nazywamy wierzchołek którego obaj sąsiedzi leżą powyżej niego i kąt wewnętrzny w wierzchołku v jest większy od $\pi\,(180^\circ)$.
Algorytm polega na schodzeniu po wierzchołkach w dół z obu stron, lewej i prawej. Jeśli najniższy punkt jest taki sam dla obu ze stron to oznacza, że w wielokącie nie ma wierzchołków dzielących i łączących.

Algorytm składa się z następujących kroków:

1. Znajdujemy punkt o największej współrzędnej y, jeśli kilka punktów ma taką samą współrzędną y, to wybieramy ten o największej współrzędnej x.
2. Przechodzimy po punktach odwrotnie do kierunku wskazówek zegara zaczynając od najwyższego, aż znajdziemy taki punkt p, że następny ma nie mniejszą współrzędną y niż p.
3. Przechodzimy po punktach w kierunku wskazówek zegara zaczynając od najwyższego, aż znajdziemy taki punkt p', że następny ma nie mniejszą współrzędną y od p'.
4. Jeśli p jest równe p', to wielokąt jest y-monotoniczny.



Rysunek 3.1.1. Wielokąt określony przez algorytm jako y-monotoniczny



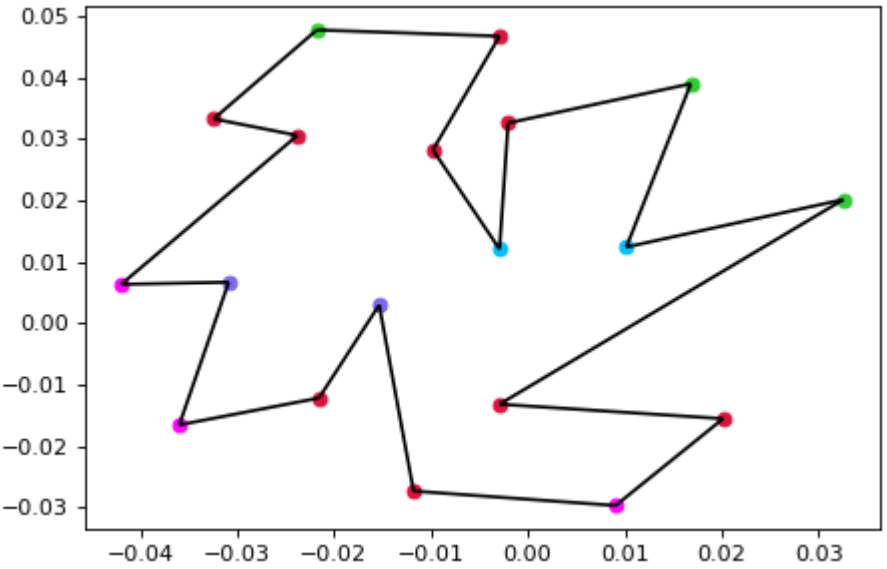
Rysunek 3.1.2. Wielokąt określony przez algorytm jako nie y-monotoniczny

3.2. Algorytm klasyfikujący wierzchołki

Każdy z wierzchołków można zaklasyfikować do jednego ze zbiorów:

- wierzchołki początkowe - obaj sąsiedzi leżą poniżej, kąt wewnętrzny przy wierzchołku < π. Zbiór oznaczony kolorem **zielonym**
- wierzchołki końcowe - obaj sąsiedzi leżą powyżej, kąt wewnętrzny przy wierzchołku < π. Zbiór oznaczony kolorem **różowym**
- wierzchołki łączące - obaj sąsiedzi leżą powyżej, kąt wewnętrzny przy wierzchołku > π. Zbiór oznaczony kolorem **niebieskim**
- wierzchołki dzielące - obaj sąsiedzi leżą poniżej, kąt wewnętrzny przy wierzchołku > π. Zbiór oznaczamy kolorem **lawendowym**
- wierzchołki prawidłowe - w pozostałych przypadkach. Zbiór oznaczamy kolorem **czerwonym**

Algorytm przechodzi po każdym wierzchołku i na podstawie wysokości sąsiadów i kąta jaki z nimi tworzy, dopasowuje go do odpowiedniego zbioru



Rysunek. 3.2.1 Przykładowa klasyfikacja wierzchołków wielokąta

3.3. Algorytm triangulacji wielokątów

Algorytm składa się z następujących kroków:

1. Sprawdzamy czy wielokąt jest y-monotoniczny
2. Znajdujemy wierzchołek o największej współrzędnej y -> h, oraz wierzchołek o najmniejszej współrzędnej y -> l
3. Przypisujemy innym wierzchołkom stronę do której należą (lewa, prawa), przechodząc po nich od góry, i dodajemy je do kolejki. Kolejka to posortowana lista punktów.
4. Wkładamy wierzchołek h i pierwszy z kolejki na stos.
5. Przeglądamy resztę wierzchołków z kolejki:

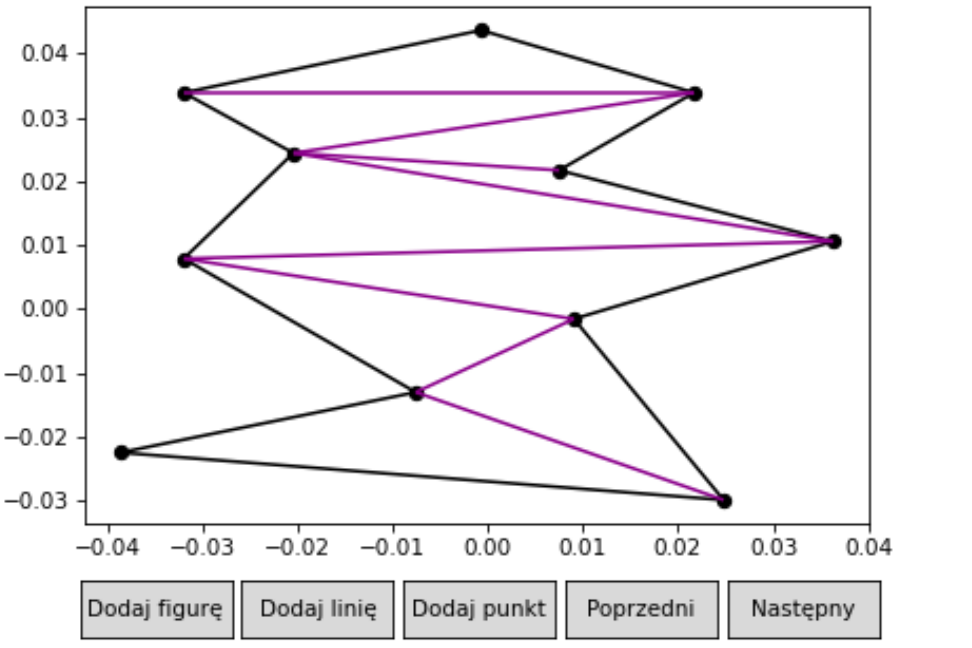
a) jeśli kolejny wierzchołek należy do innej strony niż wierzchołek ze szczytu stosu, to łączymy go ze wszystkimi wierzchołkami ze stosu, z którymi nie tworzy boku wielokąta. Na stosie pozostawiamy wierzchołek ze szczytu stosu i obecnie przetwarzany punkt.

b) jeśli kolejny wierzchołek nie należy do tej samej storny co wierzchołek ze szczytu stosu to analizujemy trójkąty utworzone przez przetwarzany wierzchołek z wierzchołkami zdejmowanymi ze stosu - tak długo jak trójkąty należą do wielokąta, łączymy ich wierzchołki. Na stosie zostają wierzchołki dla których nie udało się stworzyć trójkąta

6. Łączymy punkty ze stosu z wierzchołkiem l (jeśli nie są już połączone).

Do wizualizacji użyto następujących kolorów:

- czerwony - wierzchołek znajdujący się na stosie
- niebieski - wierzchołki dla których sprawdzamy, czy trójkąt należy do wielokąta
- magenta - dodane przekątne



Rysunek. 3.3.1 Przykładowa triangulacja wierzchołków

3.4. Użyte struktury danych

Do reprezentacji wielokąta użyłam tablicy krotek *points*. *Points[i]* oznacza i-ty wierzchołek (idąc w kierunku odwrotnym do ruchu wskazówek zegara) w postaci (x, y), gdzie x to współrzędna osi poziomej a y to współrzędna osi pionowej. Wybrałam taką strukturę do łatwiejszej i czytelniejszej implementacji kodu.

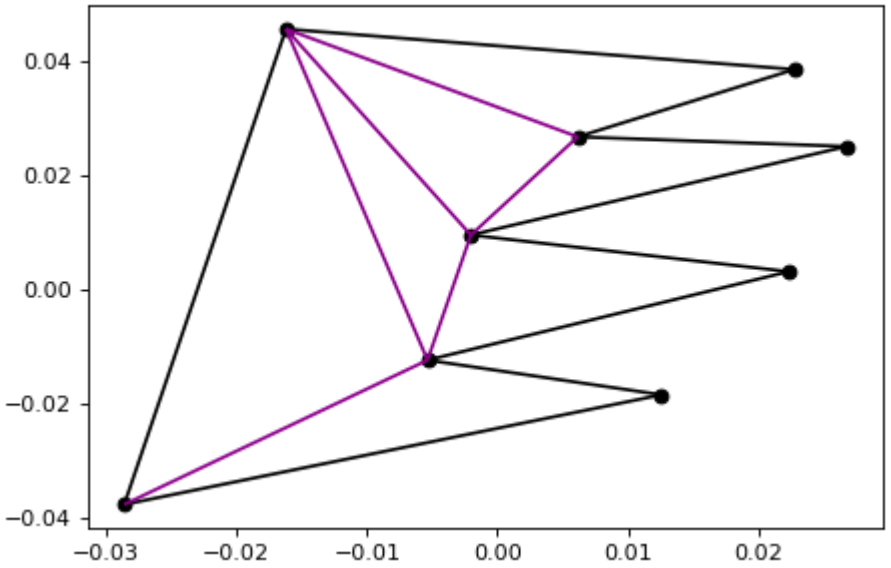
Drugą użytą przeze mnie strukturą jest klasa Vertex, która oznacza wierzchołek wielokąta i przechowuje w sobie: jego id, jego współrzędne (x, y), opis do której ‘strony’ wielokąta należy oraz współrzędne jego sąsiadów. Użyłam klasę zamiast tablicy, by mieć bardziej przejrzysty dostęp do informacji. Współrzędne sąsiadów przydają się by sprawdzić czy przekątna którą chcemy dodać nie jest bokiem wielokąta.

Wyniki triangulacji przechowuję jako listę krotek (x, y), gdzie linia pomiędzy x a y jest naszą przekątną. Taka lista jest łatwa do odtworzenia w wizualizacji

Algorytm triangulacji zwraca: listę krotek *lines*, która zawiera w sobie krotki współrzędnych wierzchołków które tworzą odcinek w triangulacji, jest ona potrzebna do odpowiedniego wyświetlenia wielokąta w wizualizacji, tablicę *scenes* która odpowiada za wizualizację kolejnych kroków algorytmu oraz listę *all_lines*, która zawiera w sobie krotki z indeksami wierzchołków które tworzą wszystkie odcinki po triangulacji (boki i odcinki dodane przez algorytm).

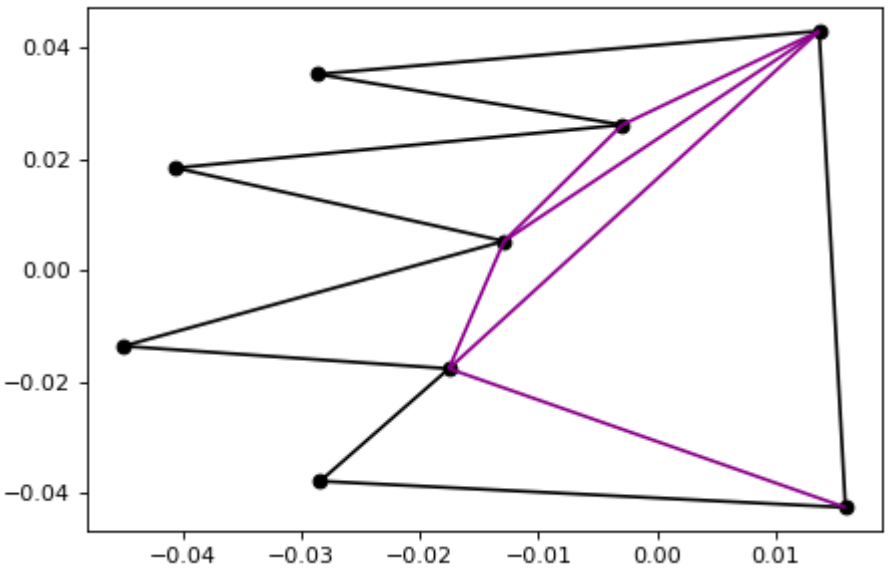
3.5. Testowanie algorytmów dla różnych zestawów danych

Wszystkie dane zostały pobrane z ręcznego narzędzia graficznego. Ważne, by kolejne wierzchołki wielokąta rysować odwrotnie do kierunku ruchu wskazówek zegara!



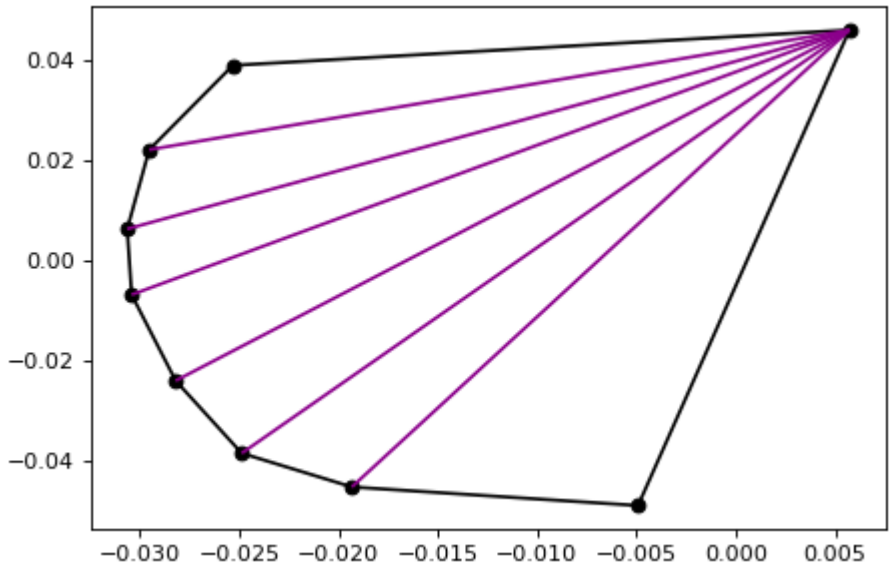
Rysunek. 3.5.1 Triangulacja dla wielokąta nr.1

Dany wielokąt został wybrany, by sprawdzić, czy łączenie punktów leżących po tej samej stronie jest poprawne (tutaj punkt najwyższy należy do obu)



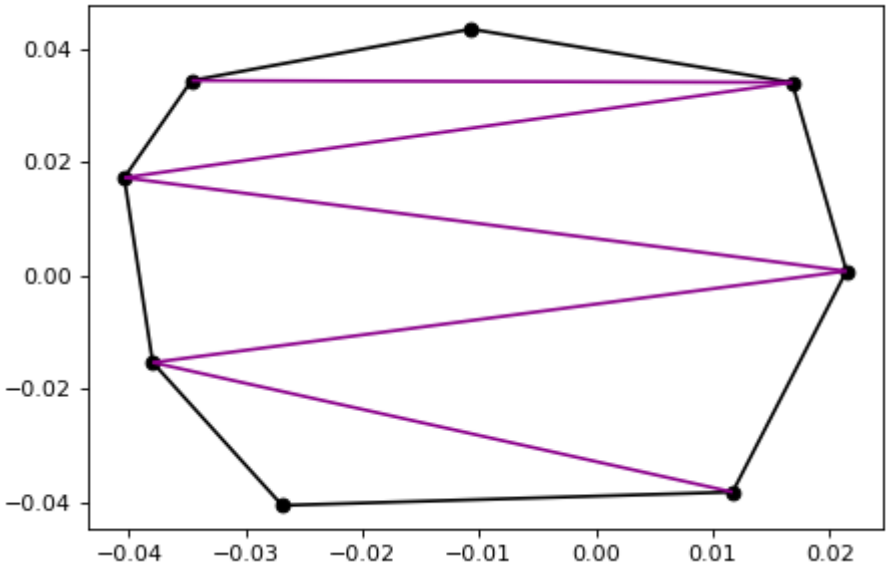
Rysunek. 3.5.2 Triangulacja dla wielokąta nr.2

Wielokąt nr. 2 spełnia tą samą sprawdzającą rolę co wielokąt nr.2. Dodatkowo, jako że jest “odwrócony” w drugą stronę, sprawdzamy czy “przechodzenie” drugą stroną jest poprawne.



Rysunek. 3.5.3 Triangulacja dla wielokąta nr.3

Powyższym wielokątem testujemy co się stanie jeśli wszystkie przekątne będą tworzone z jednym punktem i z punktami z jednej strony. Jak widać, algorytm wyznaczył triangulację poprawnie.



Rysunek. 3.5.4 Triangulacja dla wielokąta nr.4

Ten wielokąt ma na celu sprawdzenie, czy algorytm działa poprawnie, gdy przekątne powinny być utworzone tylko pomiędzy punktami należącymi do dwóch różnych stron.

4. Podsumowanie i wnioski

Testowane dane nie wykazały niepoprawności w zaimplementowanych algorytmach.