Algorytmy geometryczne

Sprawozdanie z laboratorium 3.

Dawid Zawiślak gr. Czw. 13:00 A

Dane techniczne urządzenia oraz narzędzia za pomocą których wykonano ćwiczenia:

- Laptop z systemem operacyjnym Windows 11
- Procesor Intel Pentium Gold 8505
- RAM 8 GB

Do realizacji ćwiczenia użyto środowiska Jupyter Notebook z Pythonem w wersji 3.9, wykorzystując bibliotekę visualizer przygotowaną przez KN BIT (do tworzenia wykresów), oraz matplotlib (obsługa myszki i rysowanie wielokątów zadanych przez użytkownika).

Opis ćwiczenia:

Naszym celem na laboratorium numer 3 było zaimplementowanie 3 algorytmów:

- algorytm sprawdzający czy dany wielokąt jest y-monotoniczny
- algorytm klasyfikujący wierzchołki
- algorytm triangulacji wielokąta y-monotonicznego

Należało również zaimplementować funkcjonalność zadawania nowych wielokątów za pomocą myszki oraz ich zapis i odczyt do plików.

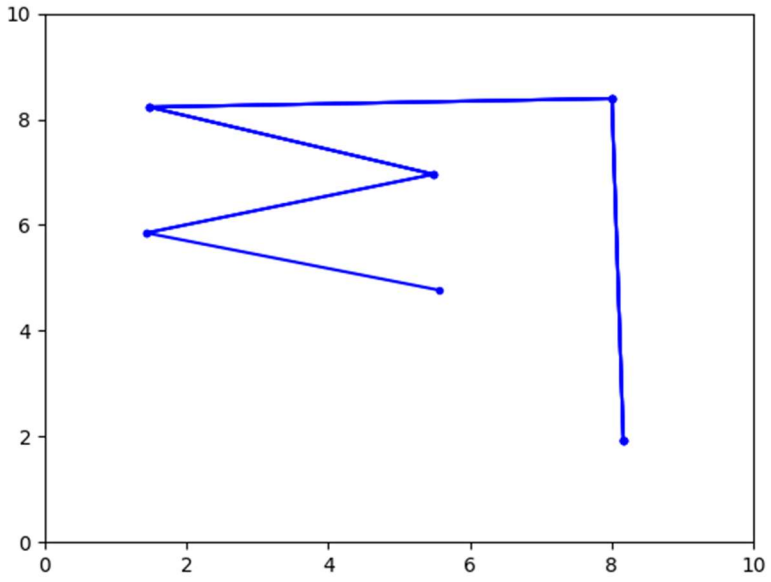
Wykonanie ćwiczenia

1) Wczytywanie punktów zadanych przez użytkownika

Za pomocą zaimplementowanego narzędzia użytkownik może wskazywać kolejne punkty wielokąta za pomocą myszki klikając w odpowiednie miejsca na wykresie (powinny one być podawane w kolejności przeciwnej do ruchu wskazówek zegara), a następnie może wygenerowany wielokąt używać do obliczeń i/lub zapisać do plików (funkcja **save_polygon_to_file**) . Można również wczytywać wcześniej zapisane z plików (funkcja **read_polygon_from_file**).

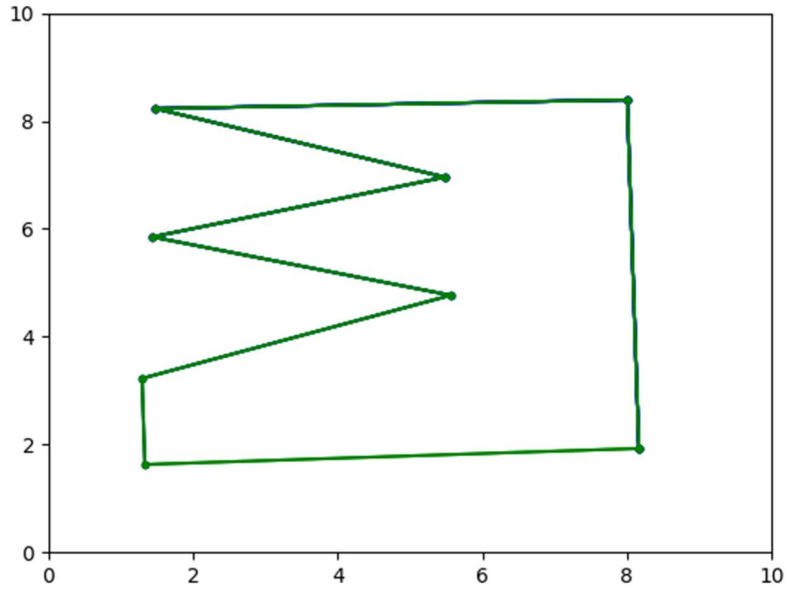
Realizowane jest to w **funkcji wczytującej wielokąt od użytkownika** znajdującej się w sekcji **Wczytywanie wielokątów** w jupyter notebook'u.

Po uruchomieniu skryptu możemy zacząć klikać myszką w miejsca gdzie chcemy dodać nowe wierzchołki:



Rysunek 1 Proces generowania wielokąta [1/2]

Wprowadzanie kończy się jak z określoną dokładnością klikniemy z powrotem w wierzchołek startowy i tym samym zamkniemy wielokąt, a zadany wielokąt dostępy jest w globalnej zmiennej **given_polygon**:

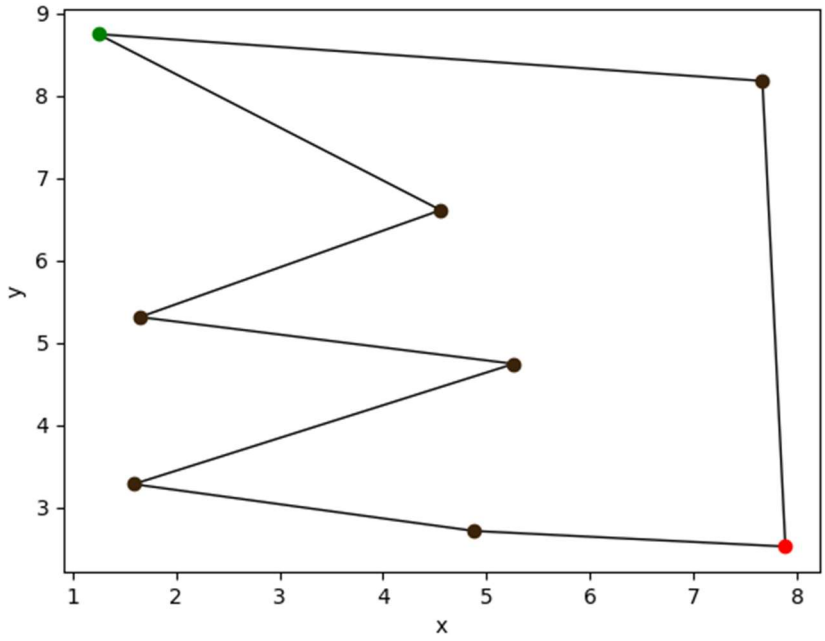


Rysunek 2 Proces generowania wielokąta [2/2]

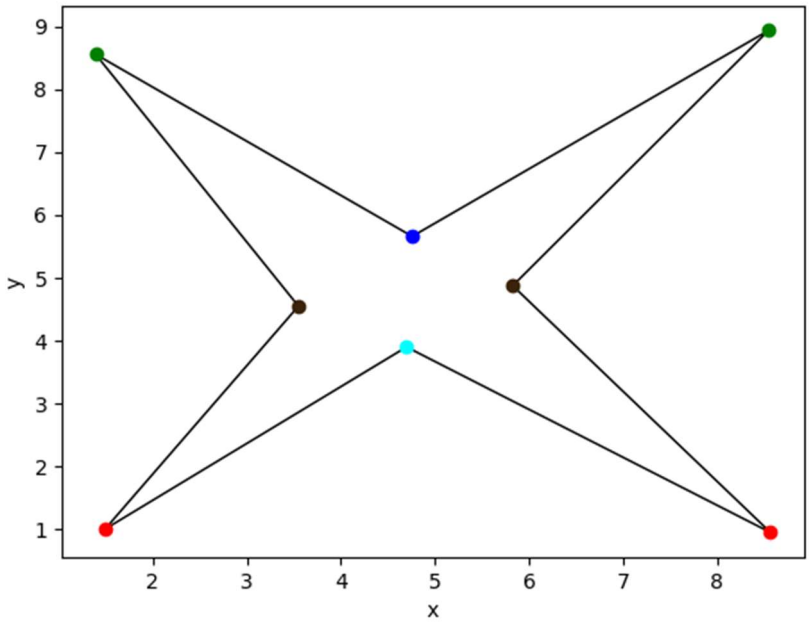
2) Sprawdzanie czy wielokąt jest y-monotoniczny

Powyższe zagadnienie jest realizowane za pomocą funkcji **is_y_monotonic.** Algorytm polaga na tym, że przechodzimy po wszystkich wierzchołkach wielokąta przeciwnie do ruchu wskazówek zegara i dla każdego wierzchołka sprawdzamy czy ze swoimi sąsiadami nie tworzy wierzchołka łączącego lub dzielącego. Tego sprawdzania dokonuje funkcja **is_connective_or_separative** która przyjmuje trójki wierzchołków leżących obok siebie i zwraca wartość prawda/fałsz. Jest w niej wykorzystywana funkcja **orient** która zwraca wyznacznik 3x3 do określenia kąta wyznaczanego przez 3 punkty. Jeśli którykolwiek wierzchołek w wielokącie jest łączący lub dzielący główna funkcja zwraca fałsz, w przeciwnym wypadku wielokąt jest y-monotoniczny i zwracana jest prawda.

Wielokąty do testów zostały zadane za pomocą wyżej opisanego narzędzia. Algorytm zwrócił prawdę dla pierwszego wielokąta i fałsz dla drugiego.



Rysunek 3 Przykład wielokąta y-monotonicznego



Rysunek 4 Przykład wielokąta nie y-monotonicznego

3) Algorytm klasyfikujący wierzchołki wielokąta

Algorytm polega na przejściu po wszystkich wierzchołka przeciwnie do ruchu wskazówek zegara i określeniu na podstawie wysokości sąsiadów oraz kąta wewnętrznego znajdującego się przy danym wierzchołku jego typu. W implementacji została wykorzystana własna implementacja wyznacznika 3x3 (funkcja **orient**).

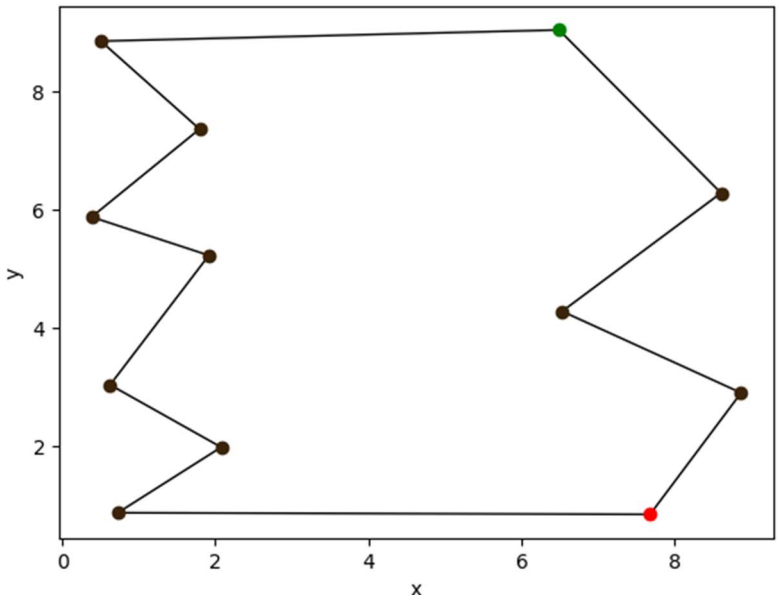
Każdy wierzchołek musi należeć do któregoś z poniższych typów:

- Początkowe - obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma mniej niż 180 stopni,
- Końcowe - obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma mniej niż 180 stopni,
- Dzielący - obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma więcej niż 180 stopni,
- Łączący - obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma więcej niż 180 stopni,
- Prawidłowy - pozostałe przypadki, jeden sąsiad powyżej drugi poniżej

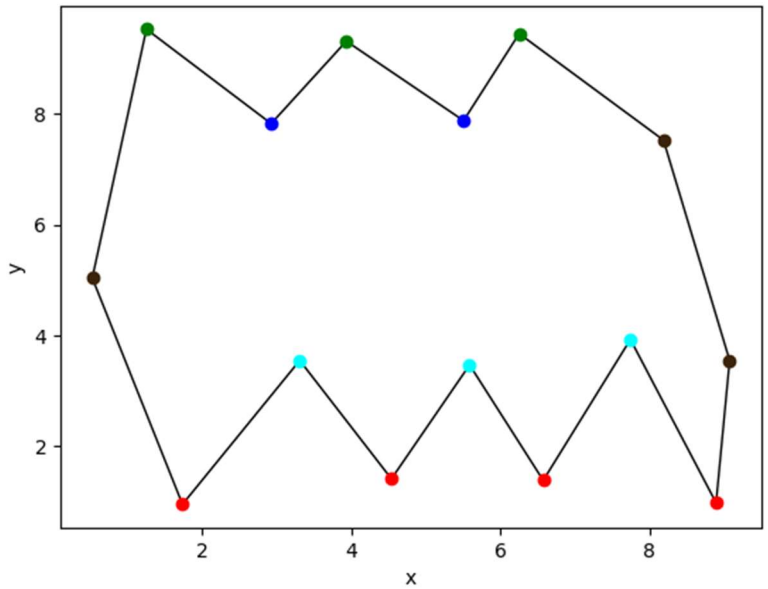
Podczas wizualizacji sklasyfikowanych wierzchołków użyto następujących kolorów:

- zielony – wierzchołki startowe
- czerwony – wierzchołki końcowe
- niebieski – wierzchołki łączące
- jasno niebieski – wierzchołki dzielące
- brązowy – wierzchołki prawidłowe

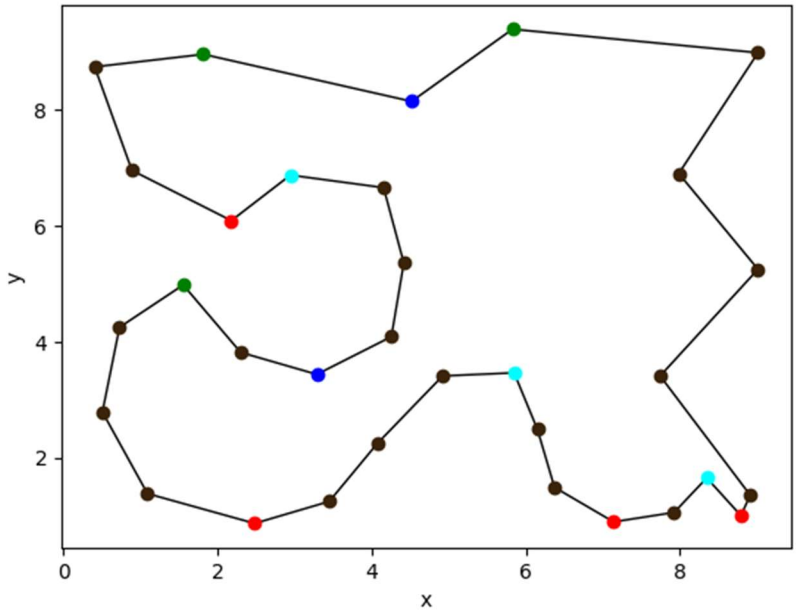
Poniżej wizualizacje kilku klasyfikacji wielokątów zadanych myszką:



Rysunek 5 Wizualizacja klasyfikacji 1



Rysunek 6 Wizualizacja klasyfikacji 2



Rysunek 7 Wizualizacja klasyfikacji 3

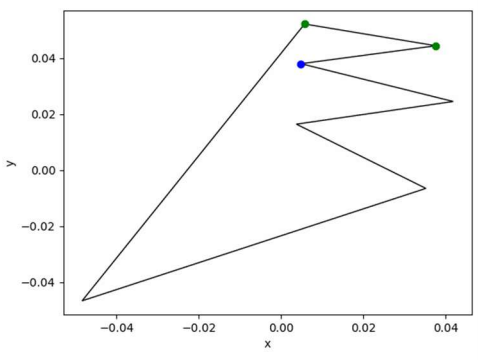
4) Algorytm wykonujący triangulacje

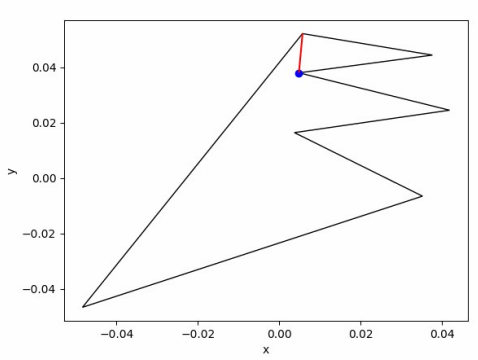
Na początku algorytm sprawdza czy wielokąt jest y-monotoniczny (funkcja **is_y_monotonic**), jeśli nie, zwracana jest pusta lista. Następnie sortuje punkty malejąco ze względu na współrzędną y używając algorytmu merge w czasie O(n) (funkcja **sort_points**) i tworzę tablice pomocniczą, która pod indeksami wierzchołków zawiera informacje o tym w którym łańcuchu (prawy/lewy) znajduje się dany wierzchołek (funkcja **get_point_side**). Wierzchołek początkowy przypisuje do łańcucha lewego, a końcowy do prawego.
Wrzucam 2 pierwsze z posortowanych wierzchołków na stos, a potem przeglądam pozostałe posortowane wierzchołki.
Możemy znaleźć się w dwóch sytuacjach:

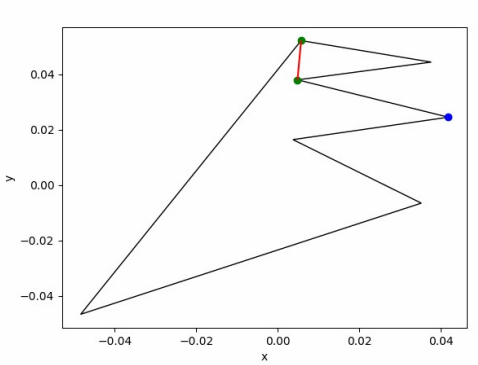
1. **Analizowany wierzchołek znajduje się w innym łańcuchu niż ten odłożony na górę stosu.** Wtedy możemy dodać przekątne pomiędzy analizowanym wierzchołkiem i każdym innym wierzchołkiem znajdującym się na stosie. Po zakończeniu procedury na stos odkładane są wierzchołek który poprzednio był na górze stosu oraz analizowany przez nas wierzchołek.
2. **Analizowany wierzchołek należy do tego samego łańcucha to ten odłożony na górę stosu.** W tej sytuacji ściągam 2 wierzchołki ze stosu i sprawdzam, czy trójkąt utworzony z nich i analizowanego wierzchołka leży wewnątrz wielokąta (funkcja **inside_polygon**). Jeśli tak dodaję przekątną pomiędzy aktualnie analizowanym wierzchołkiem i wierzchołkiem ściągniętym ze stosu jako drugi. Proces powtarzam dopóki mogę dodawać kolejne przekątne spełniając te warunki. W wypadku gdy utworzony trójkąt nie leży wewnątrz wielokąta odkładam z powrotem 2 ściągnięte wierzchołki na stos i również dodaję na stos aktualnie analizowany wierzchołek. Jeśli na stosie nie zostanie żaden wierzchołek odkładam na niego ostatnio ściągnięty wierzchołek oraz aktualnie analizowany.

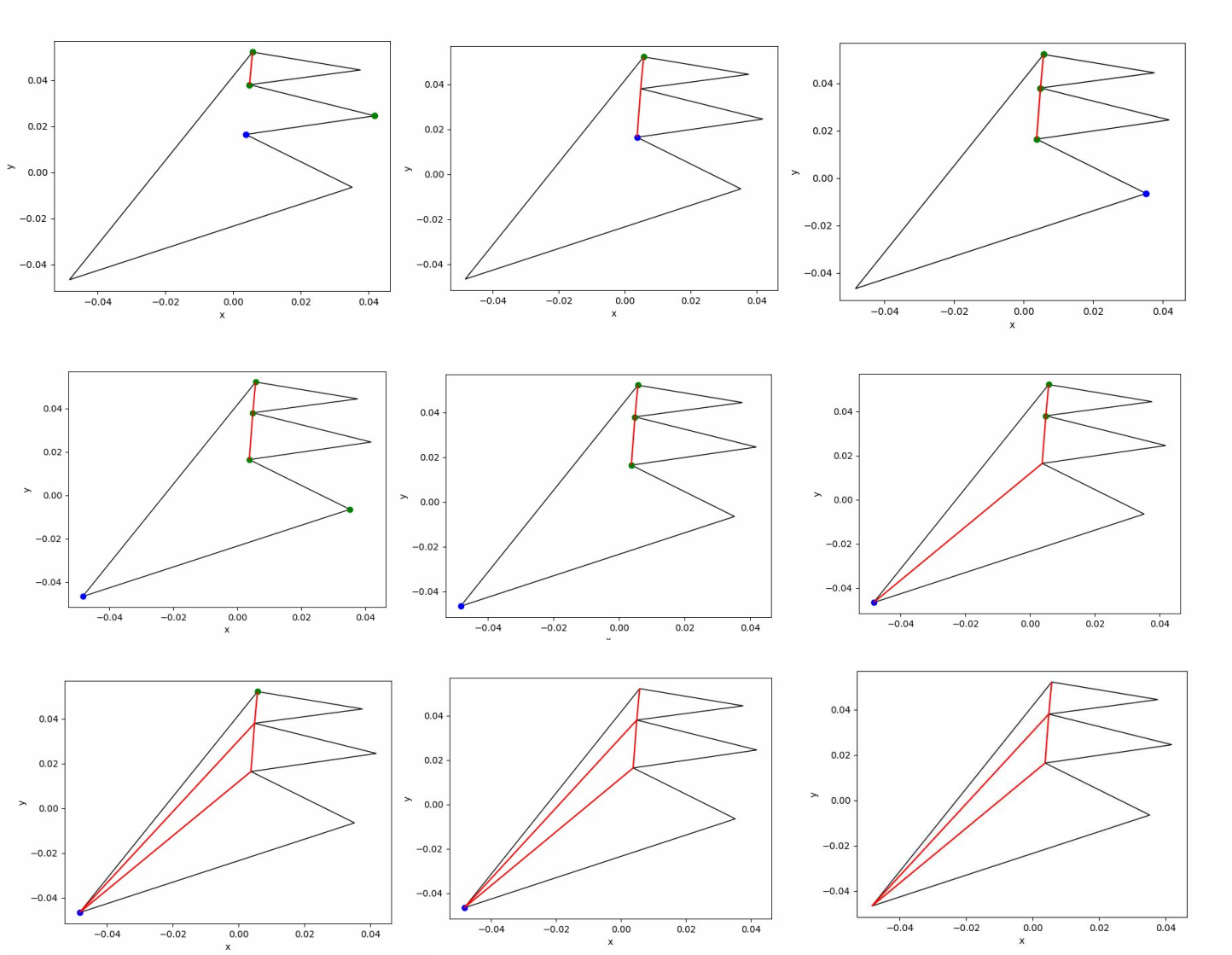
Przed dodaniem nowej krawędzi do listy krawędzi triangulacji sprawdzam czy dana krawędź nie należy do pierwotnego wielokąta (funkcja **is_neighbour**).
Funkcja zwraca listę krawędzi triangulacji.

Poniżej przedstawiony pełny proces triangulacji pewnego monotonicznego wielokąta (na zielono zaznaczono punkty znajdujące się aktualnie na stosie, a na niebiesko punkt aktualnie analizowany):

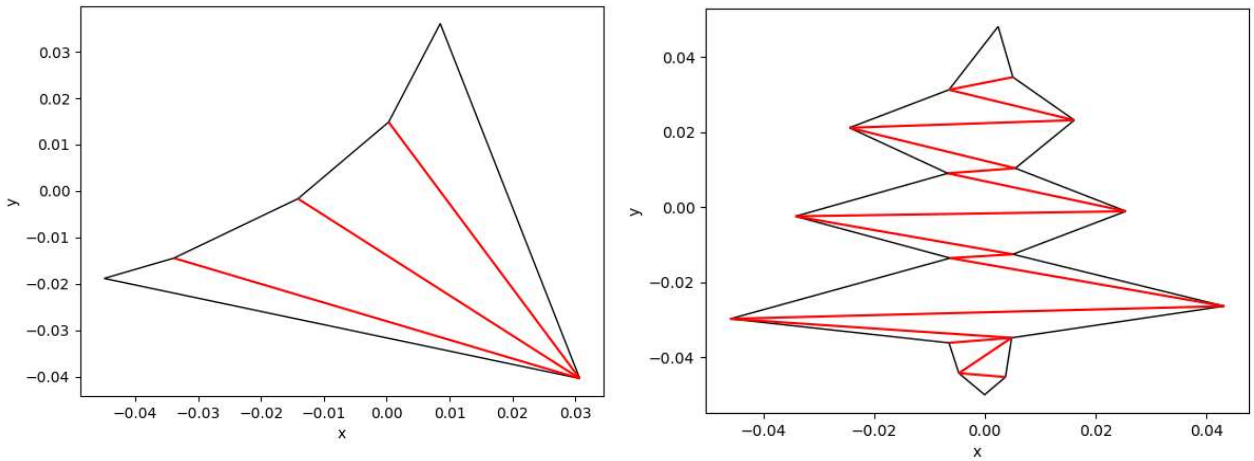


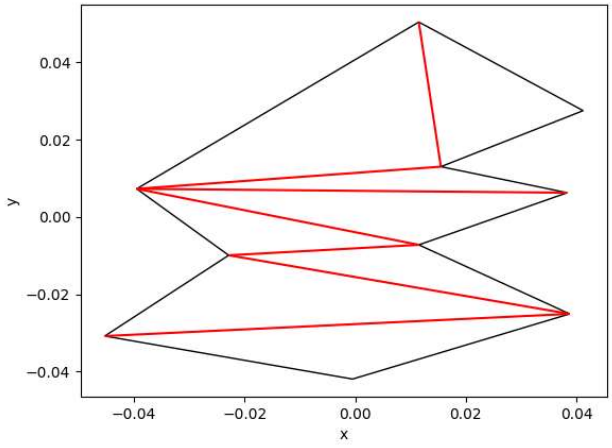


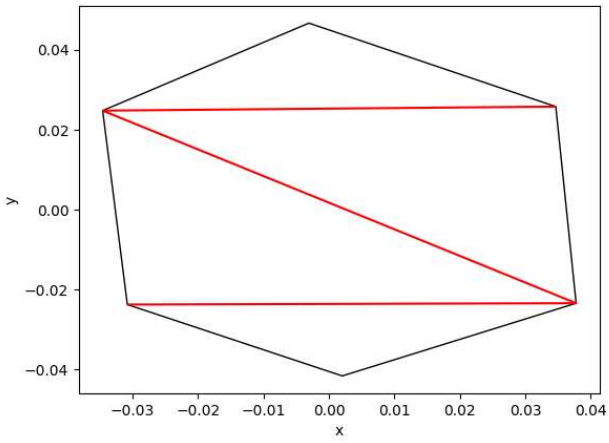


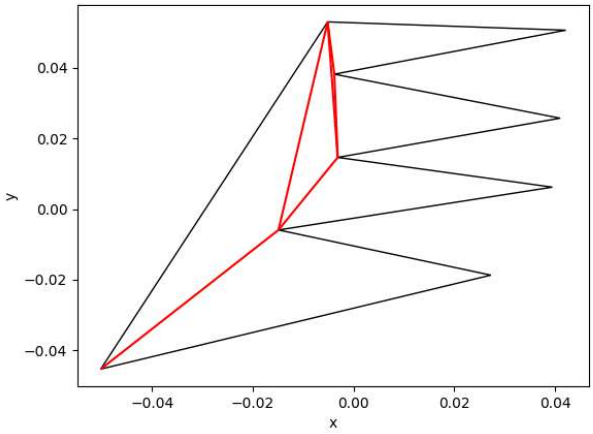


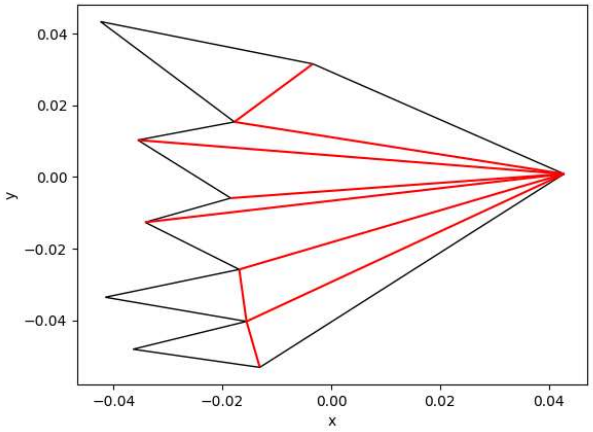
Wyniki triangulacji innych wielokątów zwrócone przez wyżej opisany algorytm:











5) Użyte struktury:

Do przechowywania wielokątów użyta została lista par współrzędnych punktów, zapisanych w kolejności przeciwnej do ruchu wskazówek zegara. Wynik triangulacji przechowywany jest jako lista trójek indeksów punktów które odpowiadają trójkątom z których składa się ztriangulowany wielokąt (jest ona dostępna w zmiennej **triangles** w funkcji **triangulation**).

Wybrałem następujący sposób, bo jest od wygodny do pracy na wyniku triangulacji, a zarazem pozwala na uniknięcie redundancji danych (przechowywanie kilkukrotnie współrzędnych punktów) oraz zajmuje stosunkowo mało miejsca (n – 2 trójkątów, na każdy po 3 liczby całkowite).

6) Wnioski:

Testy ułożone przez KN BIT uwzględniają wiele skrajnych przypadków. Algorytm zwraca poprawne wyniki dla wszystkich więc można stwierdzić, że został zaimplementowany poprawnie.