Laboratorium 3
Triangulacja Wielokątów Monotonicznych

Łukasz Kwinta

1 Dane Techniczne

Procesor: AMD Ryzen 7 5700U
System operacyjny: Ubuntu 20.04 w środowisku WSL 2 na Windows 11 x64
Pamięć ram: 32 GB DDR4

Środowisko i język: Python 3.9 + Jupyter Notebook w środowisku Anaconda
Wykresy tworzyłem przy pomocy narzędzia przygotowanego przez KN Bit, do obliczeń numerycznych używałem biblioteki numpy. Dane przechowywałem w zmiennych typu float – typ danych o rozmiarze 64 bitów, odpowiednik typu double w języku C.

2 Opis Realizacji Ćwiczenia

Celem ćwiczenia była implementacja algorytmu obliczającego triangulację wielokątów monotonicznych.

2.1 Szczegóły techniczne

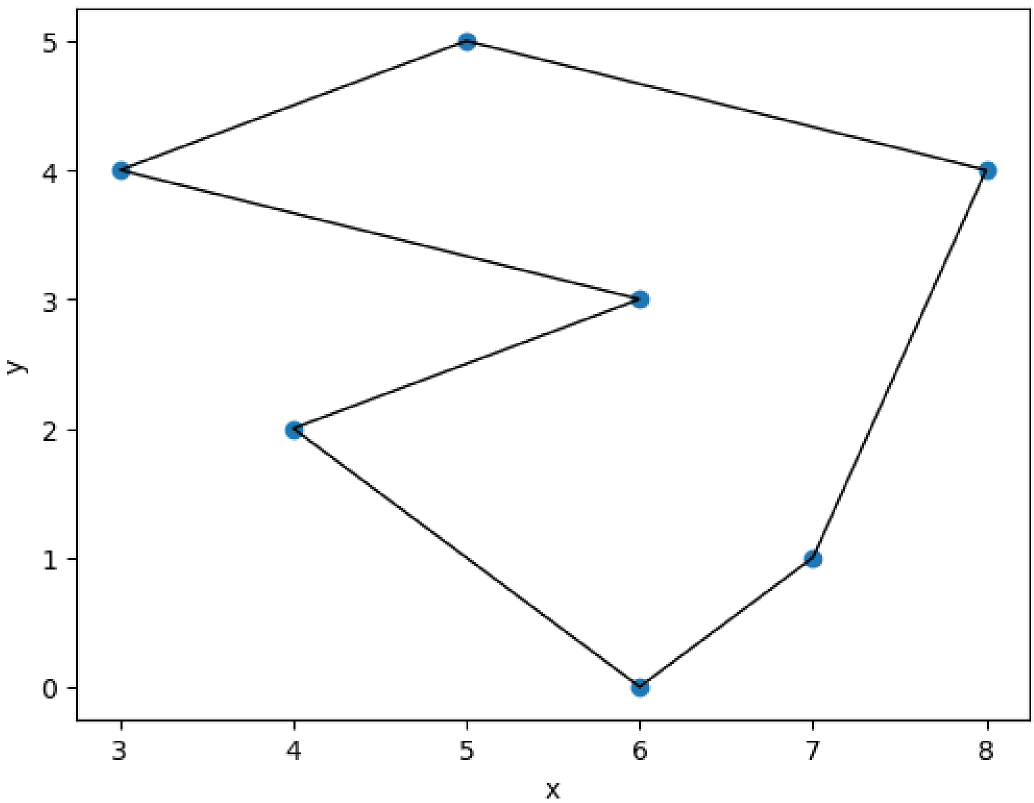
Do obliczeń używałem własnego wyznacznika 2x2 w postaci:

$$\det(a, b, c) = (b.x - a.x)(c.y - b.y) - (b.y - a.y)(c.x - b.x)$$

a dokładność zera przyjąłem jako: $\varepsilon = 10^{-14}$

2.2 Sprawdzenie y-monotoniczności

Pierwszym krokiem była implementacja funkcji sprawdzającej y-monotoniczność wielokątów - takich, które można podzielić na dwa łańcuchy w których kolejne wierzchołki wielokąta posortowane są rosnąco (jeden zgodnie ze wskazówkami zegara, drugi przeciwnie do wskazówek zegara). Przykładowy wielokąt monotoniczny przedstawiłem na Rysunku 1.



Rysunek 1: Przykładowy wielokąt y-monotoniczny

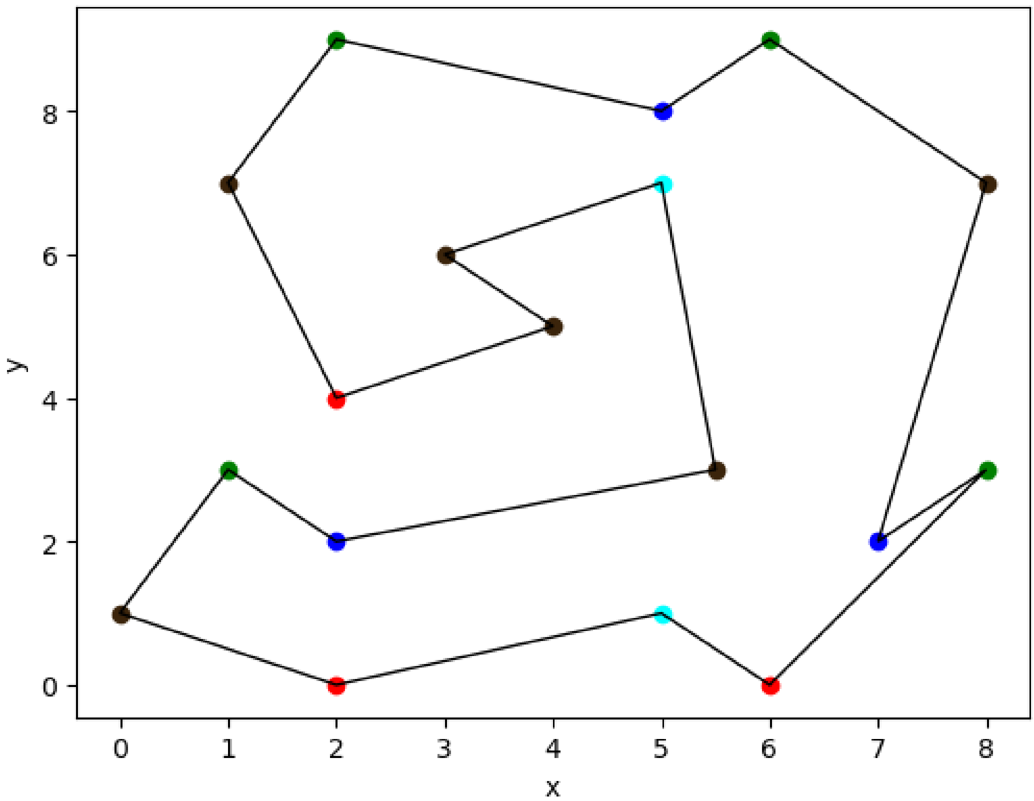
Sprawdzenia dokonałem przeglądając wierzchołki od takiego z najmniejszą współrzędną y w kierunku przeciwnym do kierunku wskazówek zegara i sprawdzam czy kolejne współrzędne rosną, a następnie po jednym punkcie zmiany monotoniczności maleją.

2.3 Klasyfikacja wierzchołków

Następną częścią ćwiczenia było zaklasyfikowanie różnych typów wierzchołków na następujące typy:

- Początkowe - oznaczone kolorem zielonym
- Końcowe - oznaczone kolorem czerwonym
- Dzielące - oznaczone kolorem cyjan
- Łączące - oznaczone kolorem niebieskim
- Prawidłowe - oznaczone kolorem brązowym

Na Rysunku 2 przedstawiam przykładowy wynik klasyfikacji wierzchołków wielokąta.



Rysunek 2: Przykładowa klasyfikacja wierzchołków

Taka klasyfikacja pozwala na podzielenie dowolnego wielokąta na wielokąty y-monotoniczne w celu triangulacji mniejszych części wielokąta prostą metodą. Powyższą metodą można też sprawdzić monotoniczność wielokąta - wielokąt monotoniczny zawierać będzie tylko wierzchołki prawidłowe oraz dokładnie jeden początkowy i jeden końcowy, lecz z pewnością jest to mniej optymalny sposób niż poprzednia metoda. Do sprawdzenia kąta wewnętrznego używałem wyznacznika 3 kolejnych punktów wielokąta.

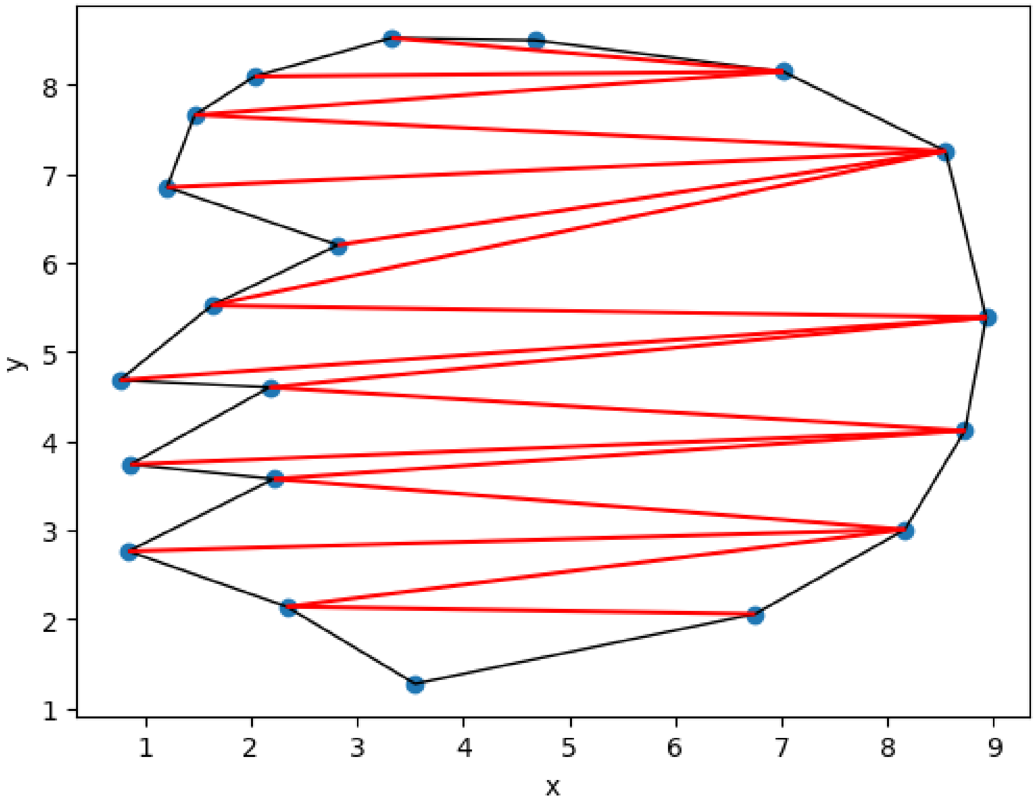
3 Triangulacja wielokąta y-monotonicznego

3.1 Opis działania algorytmu

Ostatnim i docelowym elementem ćwiczenia była implementacja algorytmu tworzącego triangulację wielokąta y-monotonicznego. Wielokąt przechowywałem jako listę kolejnych wierzchołków w kolejności przeciwnej do kierunku wskazówek zegara. Do posortowania punktów względem kierunku monotoniczności użyłem funkcji sortującej wbudowanej w listy języka Python z kluczem jako współrzędna y punktu. Zastosowałem metodę sortowania pośredniego przy pomocy tablicy indeksów pośrednich aby nie stracić informacji o początkowym ułożeniu punktów w celu łatwego sprawdzania sąsiedztwa punktów.
Aby ułatwić sprawdzanie czy nie dodaję takich samych przekątnych użyłem zbioru `set()`, opartego o hash-table zapewniającego dodawanie elementów w stałym czasie. Do określania czy sprawdzana przekątna znajduje się w czy poza wielokątem użyłem sprawdzenia odpowiedniego znaku wyznacznika punktów dla danego łańcucha.
Algorytm w postaci listy kroków przedstawia się następująco:

1. Przypisanie punktom przynależności do łańcucha
2. Posortowanie punktów wzdłuż kierunku monotoniczności
3. Włożenie dwóch pierwszych wierzchołków na stos
4. Sprawdzenie czy wierzchołek należy do tego samego łańcucha co wierzchołek na szczycie stosu
   (a) Jeśli nie to możemy dodać przekątne do wszystkich wierzchołków na stosie, a nastepnie na stos trafiają dwa ostatnio rozważane wierzchołki
   (b) Jeśli tak to badamy kolejne trójkąty składające się z przedostatniego i ostatniego wierzchołka ze stosu oraz badanego punktu:
      i. Jeśli trójkąt należy do wielokąta, to dodajemy przekątną i rozważamy kolejny trójkąt
      ii. Jeśli nie to rozważane wierzchołki dodajemy na stos

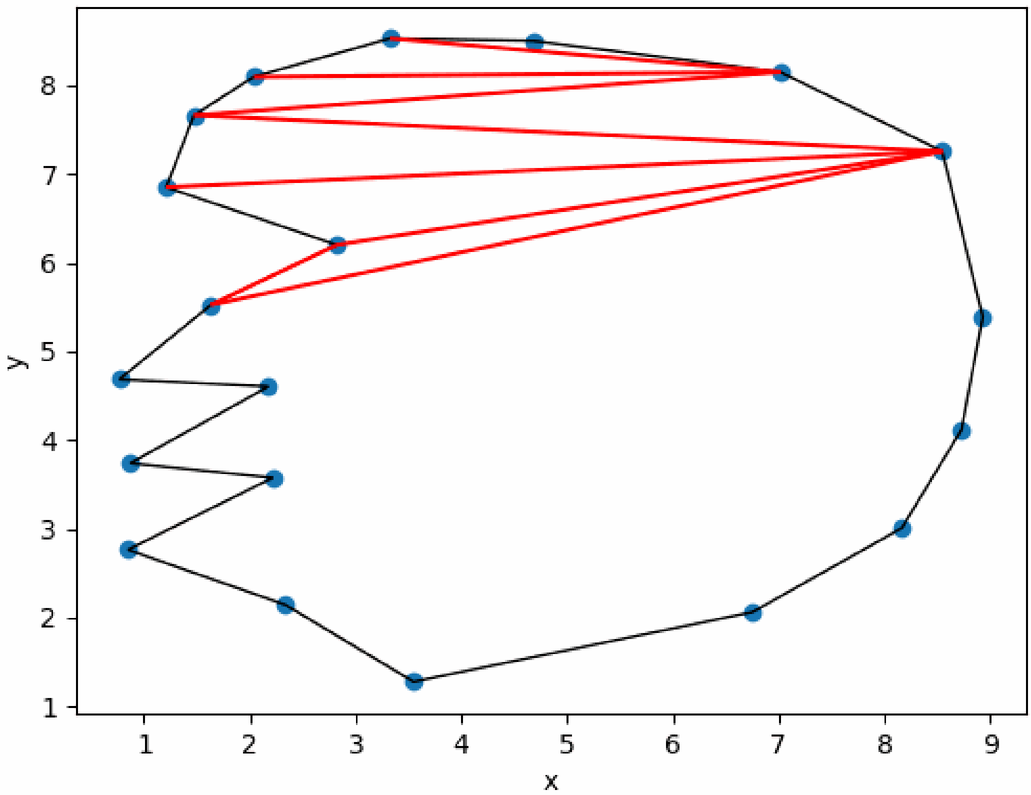
Dzięki zastosowaniu sortowania pośredniego oraz użyciu zbioru do przechowywania listy przekątnych, algorytm ma złożoność $\mathcal{O}(n \log n)$ wynikającą z sortowania wierzchołków.
Jako wynik algorytm zwraca krotki z indeksami wierzchołków między którymi znajduje się przekątna.



Rysunek 3: Przykładowy wynik działania algorytmu

3.2 Wizualizacja działania algorytmu

Dokonałem modyfikacji algorytmu aby możliwa była wizualizacja kolejnych kroków działania algorytmy, poniżej przedstawiam animację. ***Uwaga! Animacja może nie działać w każdym programie do plików PDF, została przetestowana w programie Adobe Acrobat gdzie działa. W razie gdyby animacja nie działała będzie wyświetlona jedna klatka animacji. Samą animację można znaleźć w notebooku z rozwiązaniem zadania.***



Rysunek 4: Animacja działania algorytmu na przykładowym wielokącie

4 Wielokąty testowe

4.1 Wybór wielokątów testowych

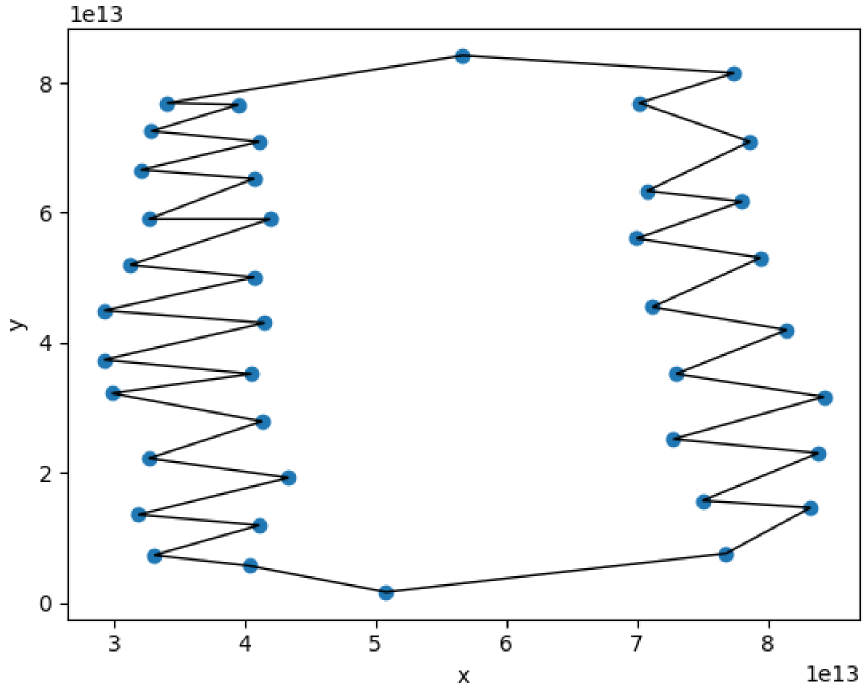
Poza testami algorytmu przygotowanymi przez KN Bit przygotowałem samemu 4 testowe wielokąty, przy pomocy zmodyfikowanego programu pozwalającemu na zadanie kolejnych punktów za pomocą myszki. Jako, że z oczywistych powodów nie mogłem testować algorytmu na dużych zbiorach danych to postanowiłem zadać wielokąty o bardzo dużych współrzędnych i bardzo małych współrzędnych gdyż w takich przypadkach dokładność liczb zmiennoprzecinkowych gra istotną rolę i mogłyby wystąpić błędy sugerujące dobranie złej dokładności zera. Jeśli chodzi o ułożenie punktów to zdecydowałem się na przetestowanie wielokątów z prostymi odcinkami i takich z bokami w "zygzak". Wybrałem takie ułożone empirycznie na podstawie doświadczeń z testami przygotowanymi przez KN Bit.

Po analizie wyników algorytmu stwierdzam, że wyniki działania algorytmu są poprawne.

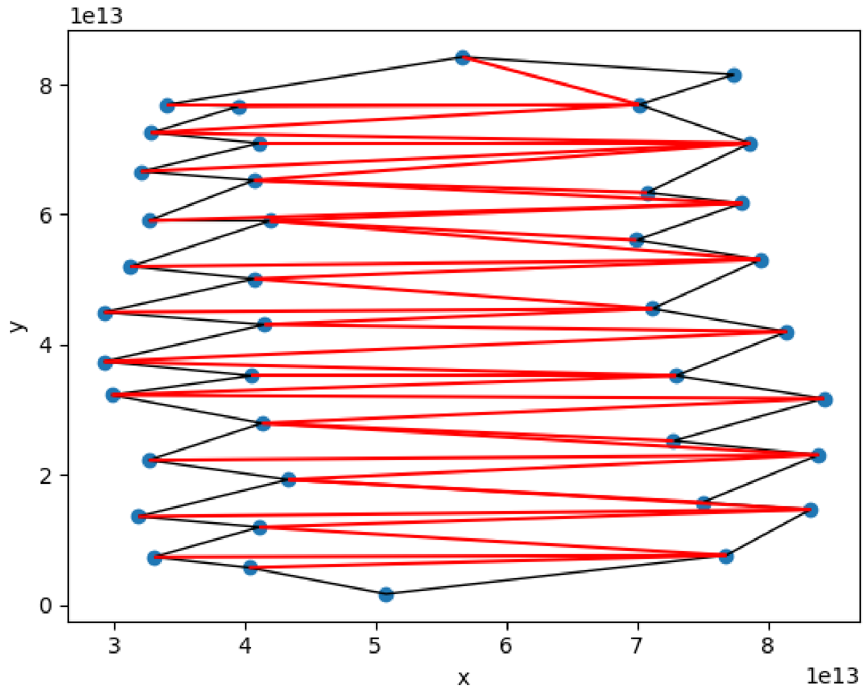
W notebooku można też obejrzeć wizualizację działania algorytmu (animację).

4.2 Wielokąt A

Pierwszy wielokąt to taki którego współrzędne są rzędu 10^{13} i zawiera wspomniane wcześniej zygzaki na bokach.



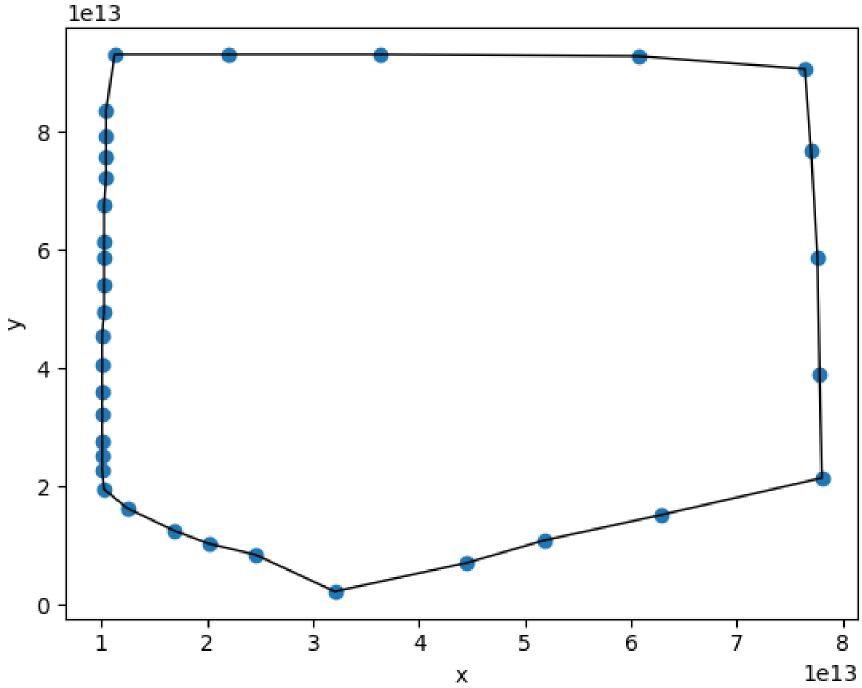
Rysunek 5: Wizualizacja wielokąta testowego A



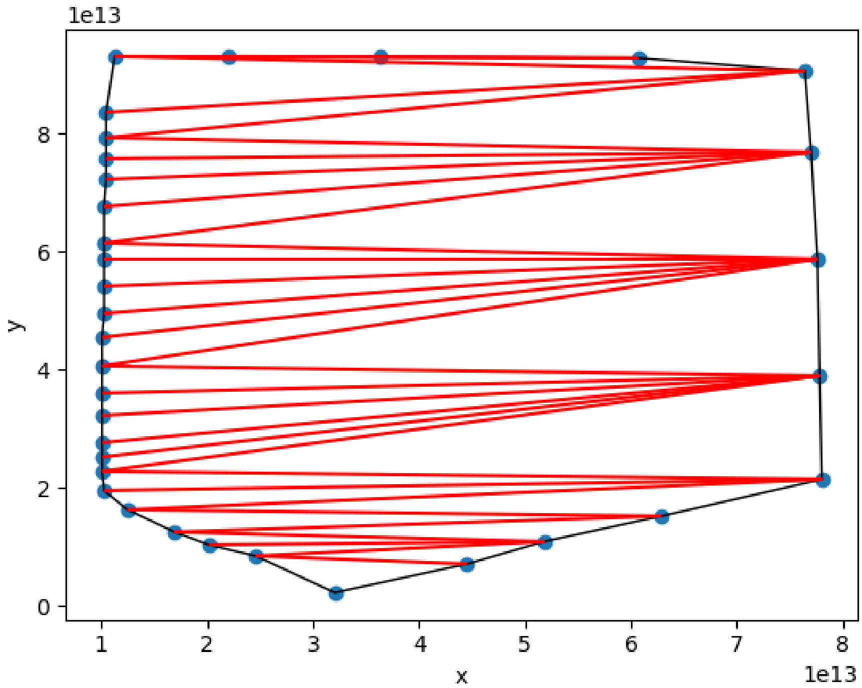
Rysunek 6: Triangulacja wielokąta A

4.3 Wielokąt B

Kolejny wielokąt to taki którego współrzędne są rzędu 10^{13} i zawiera wspomniane wcześniej linie proste



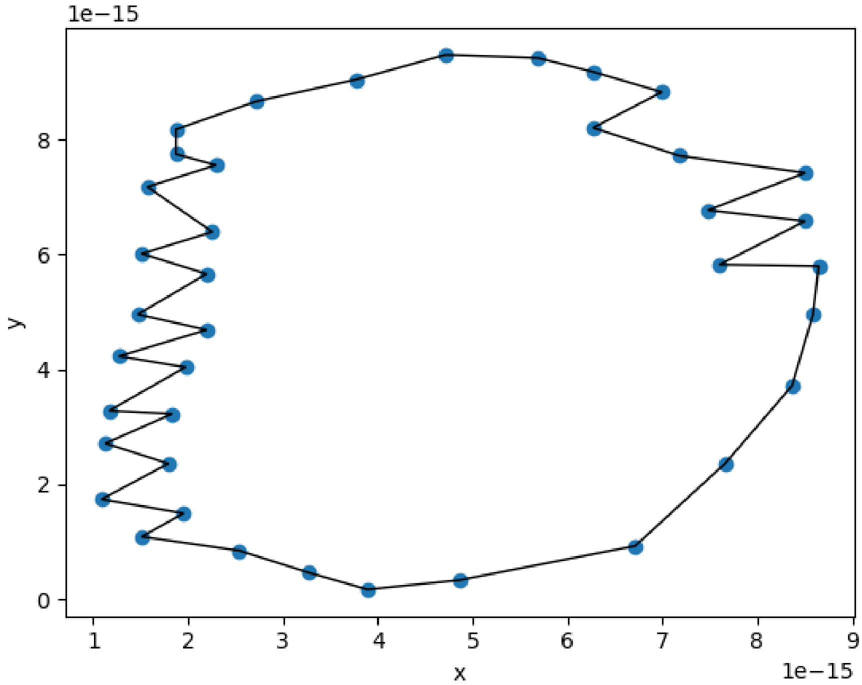
Rysunek 7: Wizualizacja wielokąta testowego B



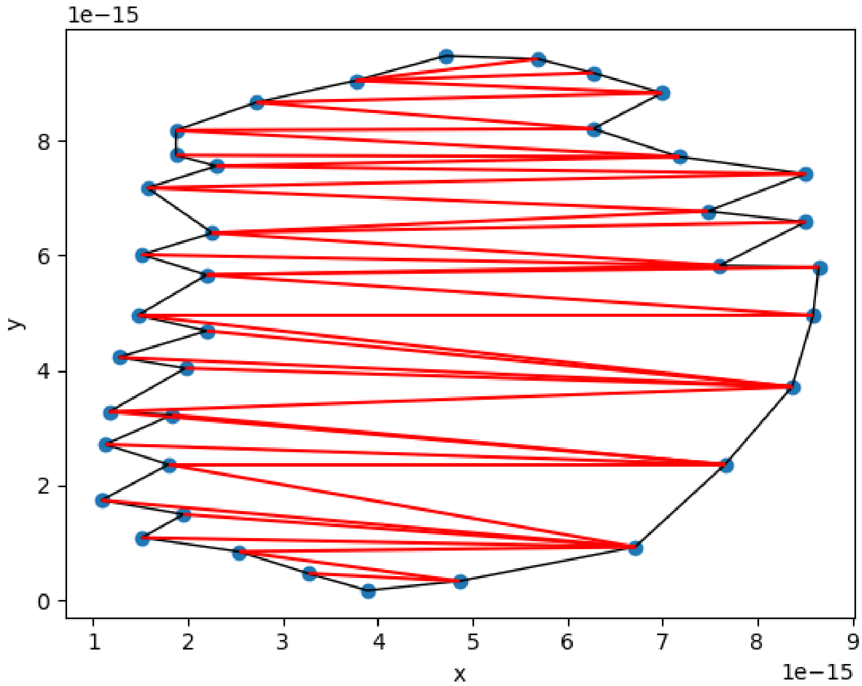
Rysunek 8: Triangulacja wielokąta B

4.4 Wielokąt C

Pierwszy wielokąt to taki którego współrzędne są rzędu 10^{-15} i zawiera wspomniane wcześniej zygzaki na bokach.



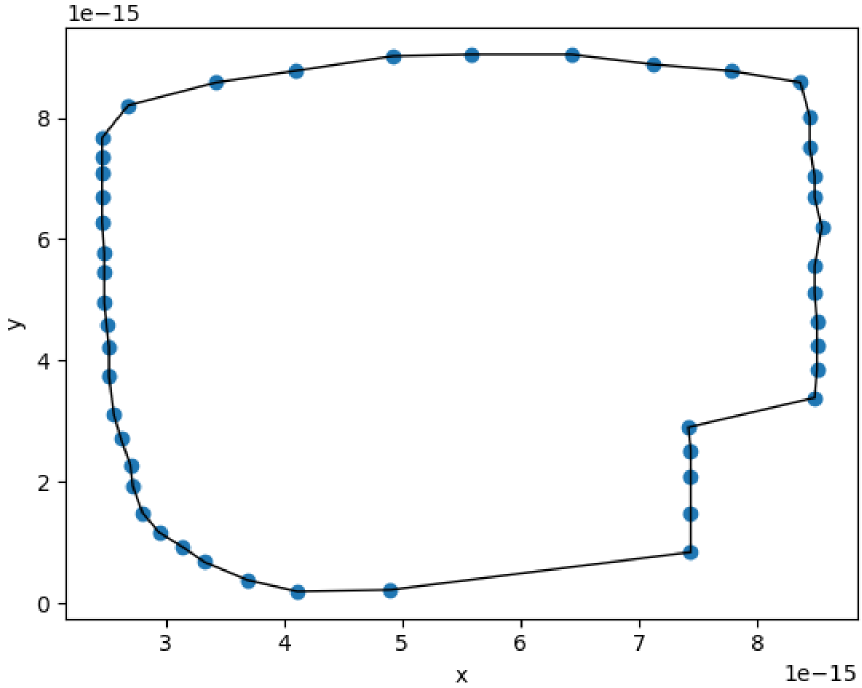
Rysunek 9: Wizualizacja wielokąta testowego C



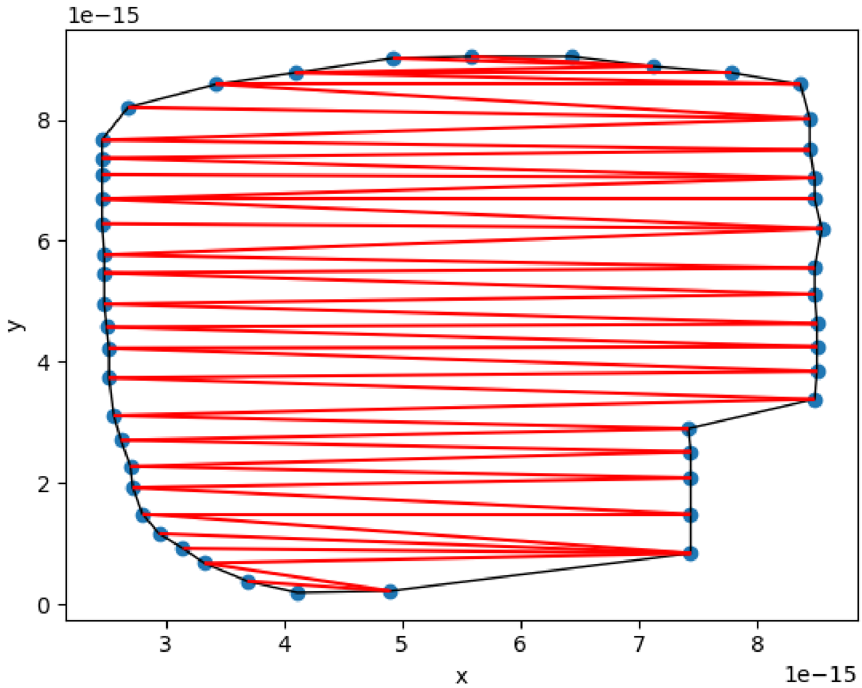
Rysunek 10: Triangulacja wielokąta C

4.5 Wielokąt D

Kolejny wielokąt to taki którego współrzędne są rzędu 10^{-15} i zawiera wspomniane wcześniej linie proste



Rysunek 11: Wizualizacja wielokąta testowego D



Rysunek 12: Triangulacja wielokąta D

5 Wnioski

Na podstawie testów przygotowanych przez KN Bit jak i moich własnych przypadków testowych można stwierdzić, że przedstawione algorytmu są zaimplementowane w prawidłowy sposób i dają prawidłowe wyniki. Najprawdopodobniej istnieją przypadki w których przyjęta dokładność zera sprawiłaby, że algorytm nie będzie działał dokładnie lecz nie udało mi się wygenerować takiego przypadku.