Algorytmy geometryczne

Anna Gut, grupa nr 7 (czwartek 16:15 A)

# Sprawozdanie z ćwiczenia nr 4 – przecinanie się odcinków

## 1.Wprowadzenie

### 1.1.Cel ćwiczenia

Celem ćwiczenia było:

- dostosowanie aplikacji graficznej tak, by była możliwość zadawania odcinków przy pomocy myszki oraz ich zapis i odczyt
- implementacja procedury generującej w losowy sposób zadaną ilość odcinków z podanego zakresu współrzędnych 2D
- implementacja algorytmu zamiatania sprawdzającego, czy co najmniej jedna para odcinków w zadanym zbiorze się przecina
- implementacja algorytmu wyznaczającego wszystkie przecięcia odcinków w zbiorze oraz zwracająca ilość wszystkich przecięć, ich współrzędne oraz odcinki, które się przecinają
- przetestowanie działania powyższych procedur na różnych zestawach danych oraz przeprowadzenie wizualizacji graficznej przebiegu algorytmów
- ocena słuszności implementacji struktur zdarzeń i stanu w procedurze

## 2.Uwagi techniczne dotyczące sprzętu

Ćwiczenie zostało przeprowadzone z wykorzystaniem sprzętu o następujących parametrach:

- oprogramowanie Windows 8.1 Pro
- procesor Intel i5
- pamięć RAM 8 GB
- 64-bitowy system operacyjny
- zainstalowane narzędzie graficzne oparte o ProjektJupyter

## 3. Opis struktur wykorzystanych do przechowywania danych

1) Sprawdzanie, czy co najmniej jedna para odcinków w zbiorze się przecina:

   Do implementacji tej procedury wykorzystana została struktura SortedSet z biblioteki sortedcontainers, porządkująca odcinki względem współrzędnej y początku odcinka. W tej strukturze przechowywany był aktualny stan miotły. Zdarzenia przechowywane były w kolejce priorytetowej typu min z biblioteki heapq – kluczem porządkującym elementy była współrzędna x danego punktu należącego do zdarzeń.

2) Wykrywanie wszystkich punktów przecięć w zbiorze odcinków:

Implementacja struktury stanu miotły przeprowadzona została w analogiczny sposób, jak w poprzednim przypadku. Jednak tym razem również zdarzenia były przechowywane przy wykorzystaniu SortedSet. Taka zmiana była konieczna do uniknięcia wielokrotnego dodania tego samego punktu przecięcia do zbioru. Nie miało to znaczenia w poprzednim algorytmie, ponieważ program wykonywał się do momentu znalezienia pierwszego punktu przecięcia. Widzimy więc, że w przypadku tych dwóch procedur nie jest konieczne obsługiwanie struktury zdarzeń w taki sam sposób.

# 4. Opis obsługi poszczególnych zdarzeń w algorytmach

Poniżej przedstawiony jest opis obsługi zdarzeń: początku odcinka, końcu odcinka oraz przecięcia odcinków dla zaimplementowanych procedur.

1) Procedura znajdywania jednego punktu przecięcia:

- zdarzenie jest początkiem odcinka – wykonujemy procedurę insert, która polega na dodaniu nowego odcinka do struktury stanu, a następnie sprawdzenie, czy istnieje przecięcie pomiędzy nim, a jego sąsiadami w tej strukturze
- zdarzenie jest końcem odcinka – wykonujemy procedurę remove, która polega na sprawdzeniu, czy istnieje przecięcie pomiędzy sąsiadami tego odcinka w strukturze stanu miotły, a następnie usunięcie odcinka z tej struktury
- zdarzenie jest przecięciem – ewentualne przecięcie pomiędzy dwoma odcinkami jest wykrywane podczas wykonywania się procedur insert oraz remove. W przypadku znalezienia punktu przecięcia, program przestaje się wykonywać

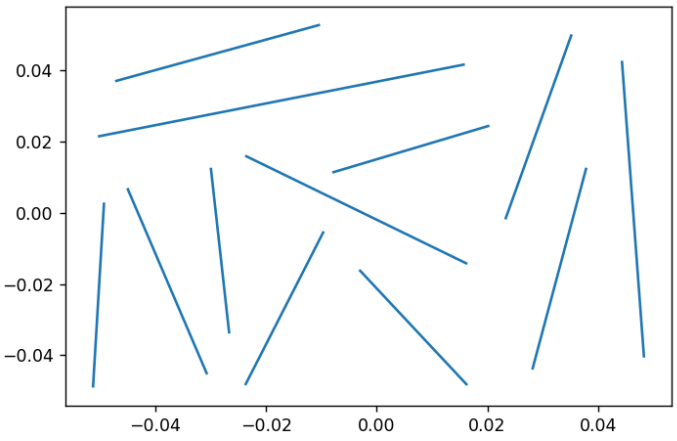
2) Procedura znajdywania wszystkich przecięć w zbiorze:

- zdarzenie jest początkiem odcinka – aktualizujemy klucz dla struktury stanu – nowym kluczem staje się współrzędna x zdarzenia, a następnie wykonujemy procedurę insert, która polega na wstawieniu odcinka do struktury stanu, a następnie na dodaniu lub usunięciu ze struktury zdarzeń ewentualnych przecięć pomiędzy odcinkiem, a jego sąsiadami w strukturze stanu
- zdarzenie jest końcem odcinka - aktualizujemy klucz dla struktury stanu – nowym kluczem staje się współrzędna x zdarzenia, a następnie wykonujemy procedurę remove, która polega na znalezieniu ewentualnych przecięć pomiędzy odcinkiem, a jego sąsiadami w strukturze stanu oraz dodanie ich do struktury zdarzeń. Następnie odcinek jest usuwany ze struktury stanu miotły.
- zdarzenie jest przecięciem – wykonywana jest procedura add_intersection, która polega na dodaniu punktu przecięcia do zbioru przecięć, a następnie zmianie kolejności przecinających się odcinków w strukturze zdarzeń poprzez uaktualnienie klucza dla tej struktury – zwiększenie jego wartości o 0.01.

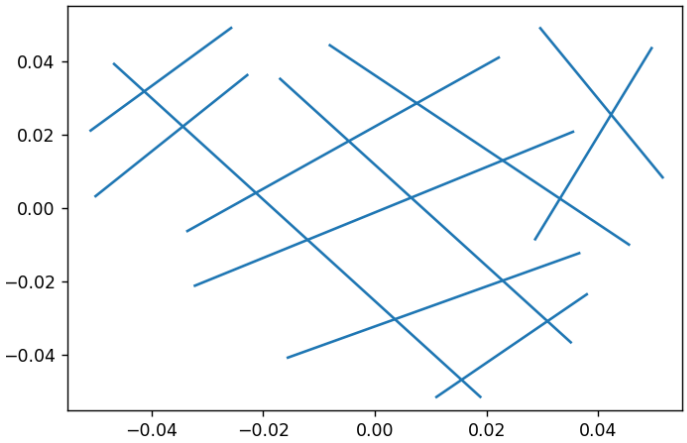
## 5.Przebieg ćwiczenia

Przy użyciu myszki zostały narysowane następujące zbiory odcinków:

Rysunek nr 1. Zbiór A

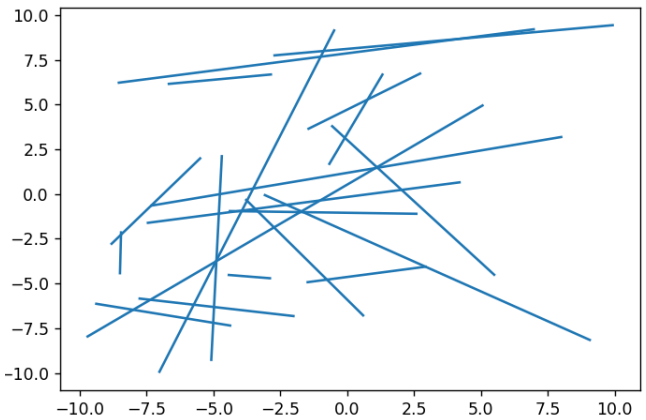



Rysunek nr 2. Zbiór B

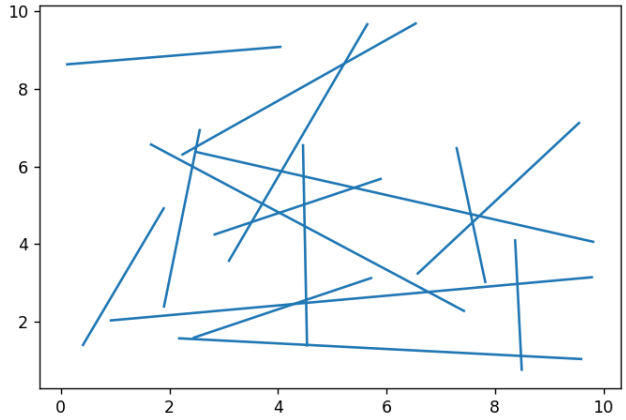



Następnie zaimplementowana została procedura generowania w losowy sposób zadanej ilości odcinków z określonego przedziału współrzędnych 2D. Poniżej przedstawione niektóre z wygenerowanych zbiorów.

Rysunek nr 3. Zbiór C




Rysunek nr 4. Zbiór D




Kolejnym krokiem było zaimplementowanie procedury sprawdzającej, czy w zadanym zbiorze odcinków znajdują się co najmniej dwa elementy takie, które się przecinają. Dla wszystkich zbiorów poza zbiorem A, na wyjściu programu otrzymywaliśmy informację, że w zbiorze istnieje punkt przecięcia dwóch odcinków.

Następnie utworzona została procedura wykrywająca wszystkie punkty przecięcia w zadanych zbiorach odcinków. Na wyjściu programu otrzymywaliśmy informację o ilości przecięć w zbiorze, wartości współrzędnych punktów przecięć oraz końców odcinków, do których te przecięcia należały.

Działaniu powyższych procedur zostały poddane powyższe zbiory. Dla obydwu algorytmów uruchomiona została również wizualizacja graficzna prezentująca ich przebieg

# 6. Klasyfikacja użytych metod

Podczas wykonywania ćwiczenia zaimplementowane zostały następujące struktury oraz funkcje:

1) Procedury pozwalające na generowanie oraz zapis i odczyt zadanych punktów:

   - **read_lines** – procedura zwracająca listę odcinków zadanych przy pomocy myszki
   - **generate_lines** – procedura zwracająca listę losowo wygenerowanych odcinków w zadanej ilości oraz zadanym przedziale współrzędnych
   - l**isttoclass** - procedura przyjmująca listę odcinków i zwracająca nową listę, w której te same odcinki przechowywane są w strukturze Line

2) Procedury pozwalające na określenie wzajemnego położenia punktów oraz odcinków:
   - **det3x3** – procedura obliczająca wyznacznik macierzy kwadratowej 3x3
   - **orient** – procedura sprawdzająca, po której stronie odcinka leży punkt

3) Struktury służące do przechowywania punktów oraz odcinków
   - **Point** – struktura przechowująca informacje o punkcie
   - **Line** – struktura przechowująca informacje o odcinku

4) Strukrury obsługujące działanie „miotły”:
   - **Sweeper** – struktura obsługująca znajdowanie jednego punktu przecięcia w zbiorze odcinków w zbiorze
   - **Sweeper1** - struktura obsługująca znajdowanie wszystkich punktów przecięcia odcinków w zbiorze

5) Procedury pozwalające na znajdywanie punktów przecięcia odcinków:

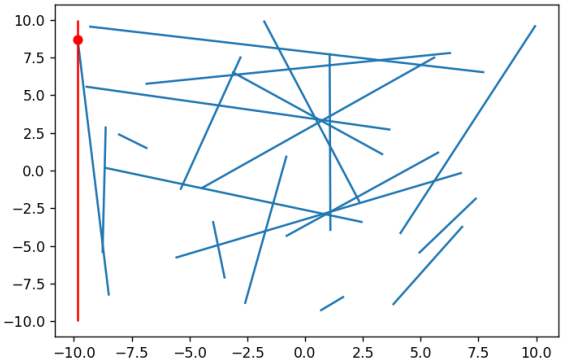
   - **is_intersection** – procedura sprawdzająca czy zadane dwa odcinki się przecinają
   - **sweep** – procedura obsługująca znajdywanie punktów przecięć poprzez przechodzenie po strukturze zdarzeń
   - **insert** – procedura dodająca nowy odcinek do struktury stanu miotły oraz sprawdzająca istnienie ewentualnych przecięć wśród odcinków sąsiadujących
   - **remove** – procedura usuwająca odcinek ze struktury stanu miotły oraz sprawdzająca istnienie ewentualnych przecięć wśród sąsiadów
   - **happened** – procedura sprawdzająca, czy dane zdarzenie jest początkiem odcinka, końcem odcinka lub punktem przecięcia dwóch odcinków
   - **line_intersection** – procedura sprawdzająca czy zadane dwa odcinki się przecinają oraz wyznaczająca punkt ich przecięcia
   - **add_intersection** – procedura obsługująca dodanie nowego punktu do zbioru punktów przecięć

# 7.Wizualizacja graficzna przebiegu algorytmów

## 1) Znajdowanie jednego punktu przecięcia

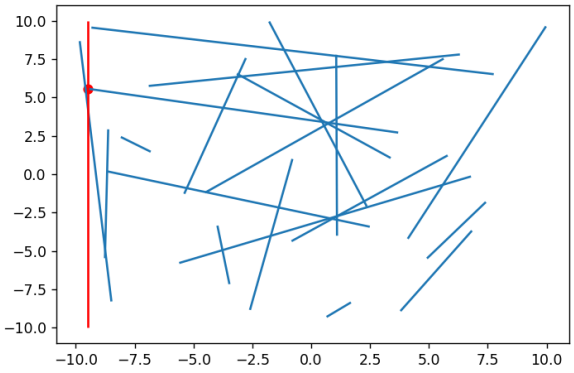
Poniżej przedstawione zostały poszczególne kroki przebiegu algorytmu wykrywającego, czy w zbiorze istnieje chociaż jeden punkt przecięcia.

Rysunek nr 5.

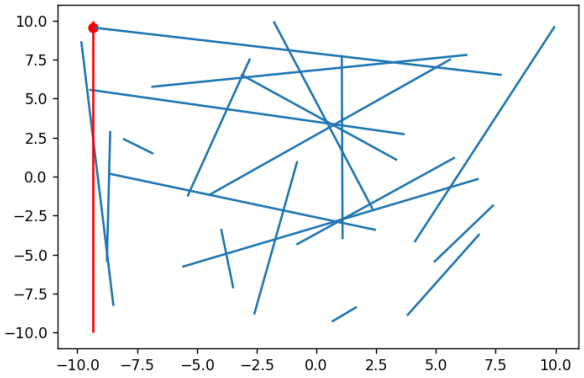



Jjjs

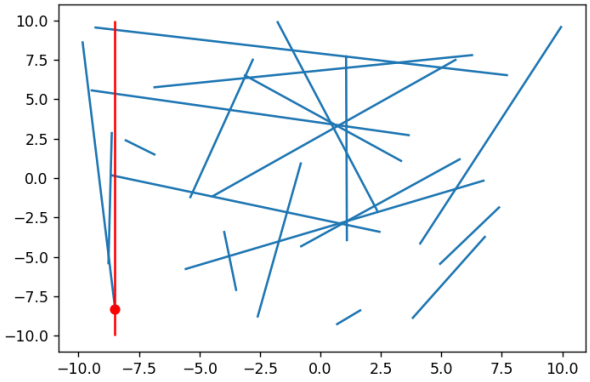
Rysunek nr 6.



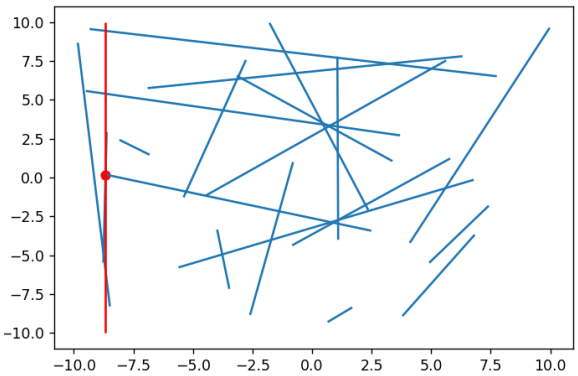

Rysunek nr 7.



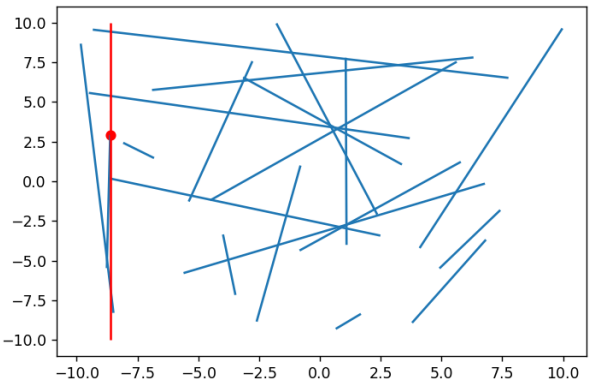

Rysunek nr 8.



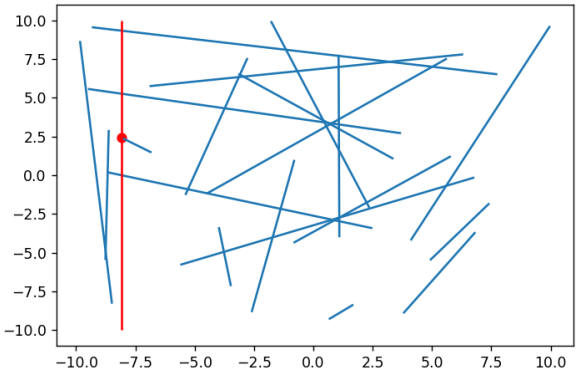

Rysunek nr 9.



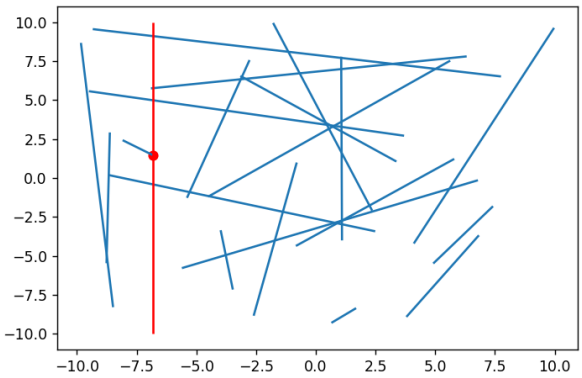

Rysunek nr 10.



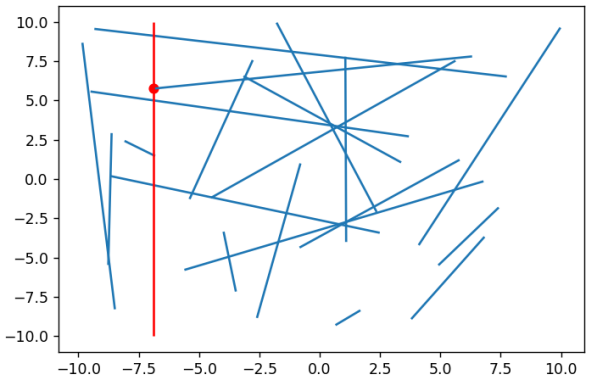
Rysunek nr 11.



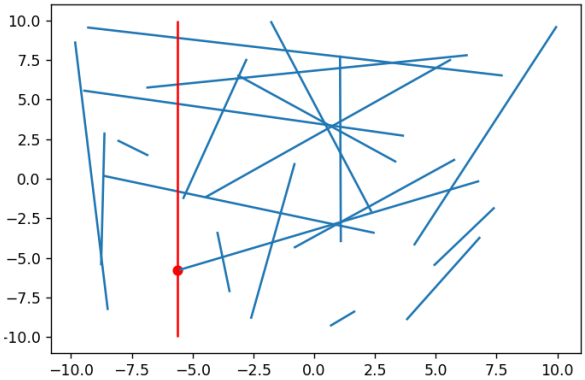

Rysunek nr 12.




Rysunek nr 13.



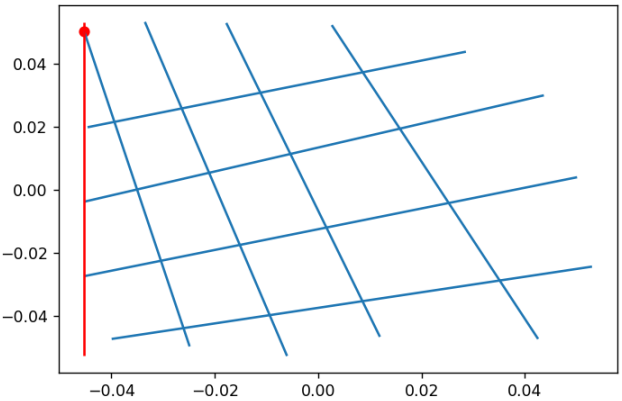

Rysunek nr 14.



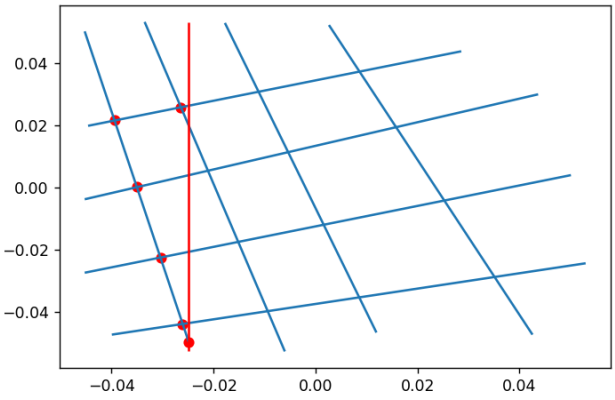

2) Znajdowanie wszystkich punktów przecięcia

Poniżej przedstawione zostały poszczególne kroki przebiegu algorytmu wykrywającego wszystkie punkty przecięcia w zbiorze B.

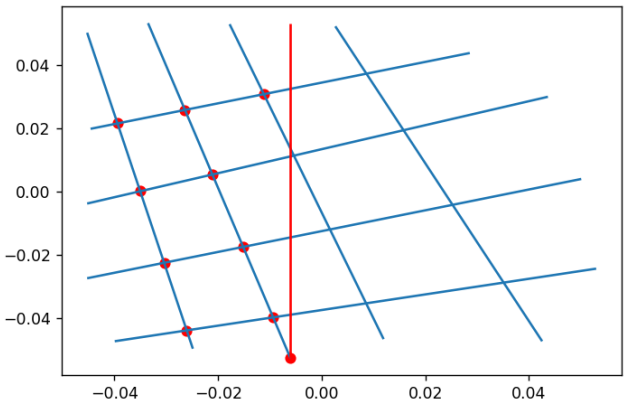
Rysunek nr 16.



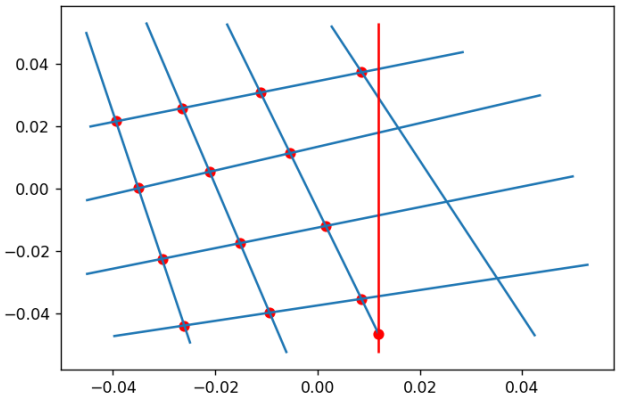

Rysunek nr 17.



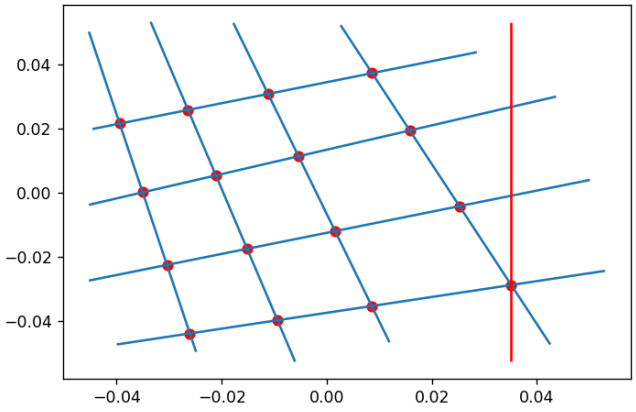
Rysunek nr 18.



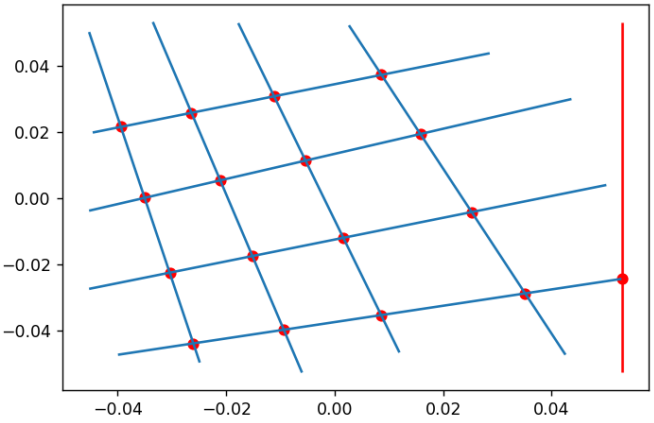

Rysunek nr 19.




Rysunek nr 20.




Rysunek nr 21.




## 8. Podsumowanie i wnioski

- analizując otrzymane wyniki oraz poszczególne kroki przebiegu procedur, można stwierdzić, że zostały one zaimplementowane w poprawny sposób. Zadane zbiory danych pozwalają na dokładne ustalenie oczekiwanego wyniku, który w każdym przypadku pokrywa się z otrzymanym na wyjściu programu
- struktury zastosowane do przechowywania zdarzeń oraz stanu miotły pozwalają na uzyskanie dobrej złożoności całego algorytmu, ponieważ pozwalają na łatwy dostęp do kolejnych elementów. Zastosowanie SortedSetu do przechowywania struktury zdarzeń w przypadku drugiego algorytmu pozwoliło na radzenie sobie z wielokrotnym wykrywaniem tego samego punktu.