Algorytmy Geometryczne
Labolatorium nr 4

Szymon Szarek

11 grudnia 2023

1 Opis ćwiczenia

Celem ćwiczenia była implementacja następujących algorytmów:

- algorytm zamiatania sprawdzający, czy choć jedna para odcinków w zadanym zbiorze się przecina
- algorytm wyznaczający wszystkie przecięcia odcinków w zadanym zbiorze

Należało również umożliwić zadawanie zbiorów odcinków w sposób graficzny za pomocą myszki.

2 Środowisko pracy

Ćwiczenie zostało napisane w języku Python za pomocą platformy Jupyter Notebook. Jako narzędzie graficzne do wizualizacji figur geometrycznych zostało użyte narzędzie przygotowane przez koło naukowe BIT, które wykorzystuje takie biblioteki jak np. *matplotlib* lub *numpy*.

Specyfikacja sprzętu:

- System Windows 11 x64
- Procesor AMD Ryzen PRO 5650U
- Pamięć RAM 16GB
- Python 3.9

3 Realizacja ćwiczenia

W ramach ćwiczenia zaimplementowane zostały wcześniej wspomniane algorytmy.

3.1 Algorytm sprawdzający, czy choć jedna para odcinków się przecina

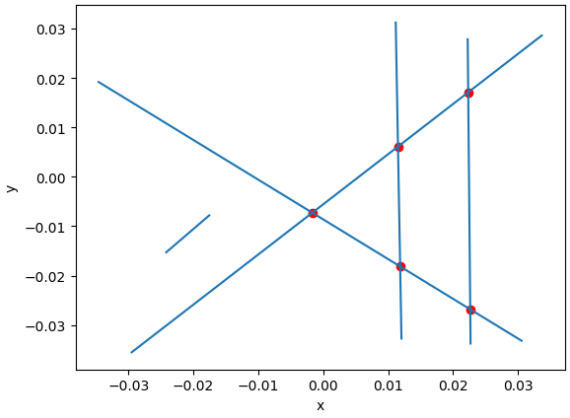
Algorytm opera się na idei zamiatania. Jako strukturę stanu wykorzystałem typ SortedSet z biblioteki *sortedcontainers*. Przechowywuje ona zbiór posortowanych elementów względem narzuconego klucza. Sprawdzanie czy element znajduje się w strukturze odbywa się w czasie $O(1)$, a usuwanie lub dodawanie elementu w czasie logarytmicznym $O(logn)$. Do struktury zdarzeń użyłem kolejkę priorytetową PriorityQueue z biblioteki queue zaimplementowaną z wykorzystaniem standardowego kopca minimum. Struktura zdarzeń przechowywuje pierwsze współrzędne (x-owe) wszystkich końców odcinków ze zbioru wejściowego. Algorytm przechodzi po kolei od najmniejszych do największych x-ów ze struktury stanu i w każdym wykonuje jedną z trzech procedur w zależności od rodzaju zdarzenia. Jeżeli jest to początek odcinka, dodajemy odcinek do struktury stanu na odpowiednie miejsce i sprawdzamy, czy istnieją przecięcia z jego sąsiadami. Jeżeli jednak zdarzeniem jest koniec odcinka, to usuwamy go ze struktury stanu i sprawdzamy, czy jego poprzedni sąsiedzi nie przecinają się ze sobą. Algorytm działa dopóki nie znajdziemy pierwszego przecięcia.

3.2 Algorytm wyznaczający wszystkie przecięcia odcinków

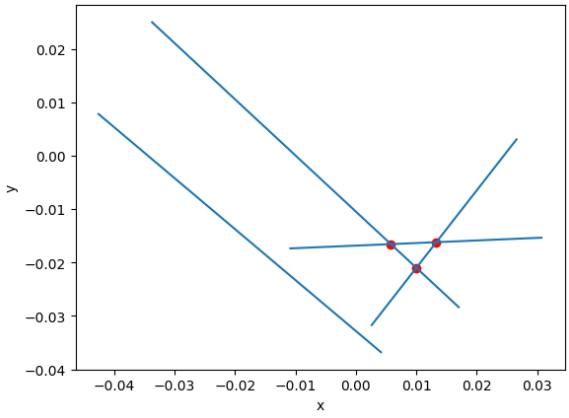
Algorytm ten działa na identycznej zasadzie co poprzedni, lecz z małymi zmianami. Interesują nas wszystkie przecięcia, więc algorytm po znalezieniu takowego nie kończy się, ale dodaje punkt przecięcia do struktury zdarzeń (jeżeli jeszcze go tam nie ma). To z kolei nasuwa nam nowy rodzaj zdarzenia do obsłużenia - punkt przecięcia. Podczas tego zdarzenia zamieniamy miejscem przecinające się odcinki w strukturze stanu i sprawdzamy ich przecięcia z nowymi sąsiadami (odpowiednio dodając je do struktury zdarzeń jeżeli istnieją i jeszcze ich tam nie ma). Do sprawdzenia, czy dany punkt przecięcia nie znajduje się już w strukturze zdarzeń wykorzystałem standardowy zbiór w języku Python z operacją sprawdzania czy element należy do zbioru w czasie stałym $O(1)$. Przechowywałem w nim pary odcinków już wykrytych przecięć. Na końcu algorytmu zbiór posłużył również do obliczenia zbióru wynikowego, który przyjmuje postać listy 3-elementowych krotek: (punkt przecięcia, indeks odcinka 1 w liście wejściowej, indeks odcinka 2 w liście wejściowej). Nie było konieczności zmiany rodzaju struktur danych w porównaniu z poprzednim algorytmem, jednak w poprzednim algorytmie nie trzeba korzystać z kolejki priorytetowej jako struktury zdarzeń. Wystarczałaby sama posortowana lista, ponieważ nie korzystamy z opcji dodawania zdarzeń do struktury na bieżąco.

4 Wyniki działania algorytmu

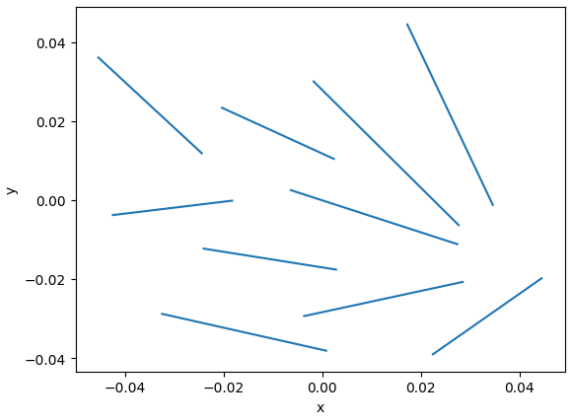
4.1 Wyznaczanie wszystkich przecięć



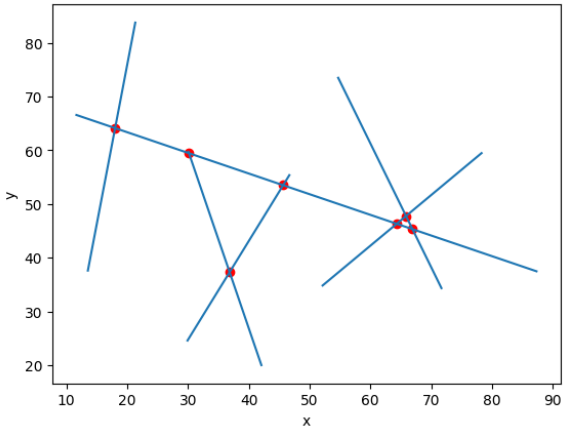
Wykres 1.



Wykres 2.



Wykres 3.



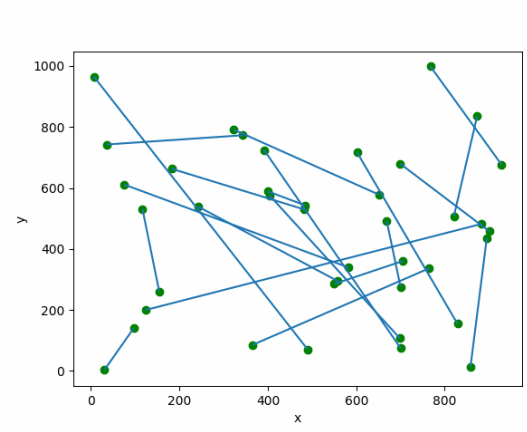
Wykres 4.

5 Wizualizacja działania algorytmów

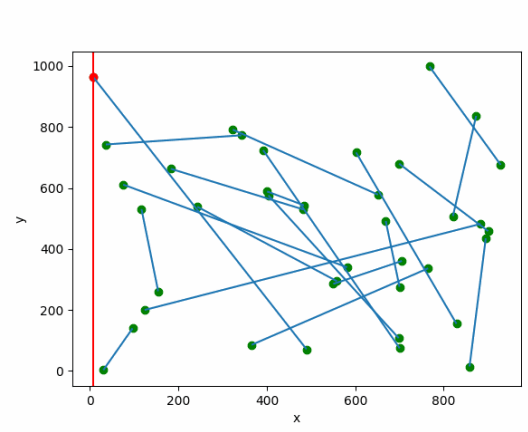
Legenda kolorów:

- Czerwony - punkt (zdarzenie) aktualnie rozpatrywany / miotła
- Zielony - początki oraz końce odcinków
- Fioletowy - punkt przecięcia

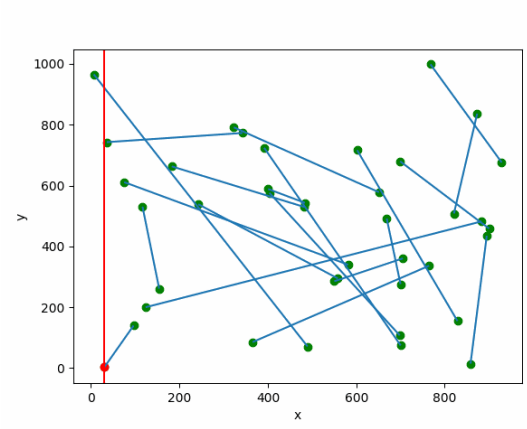
5.1 Algorytm sprawdzający, czy choć jedna para odcinków się przecina



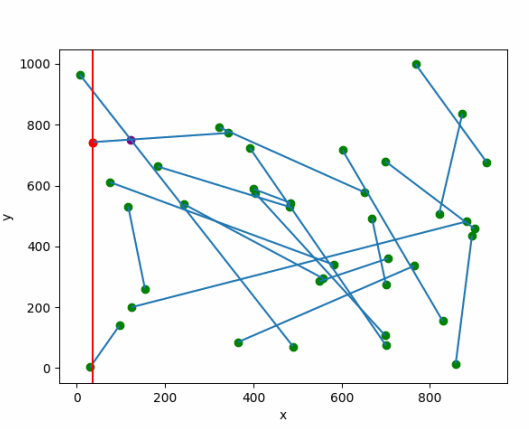
Wykres 13. Zbiór 1.



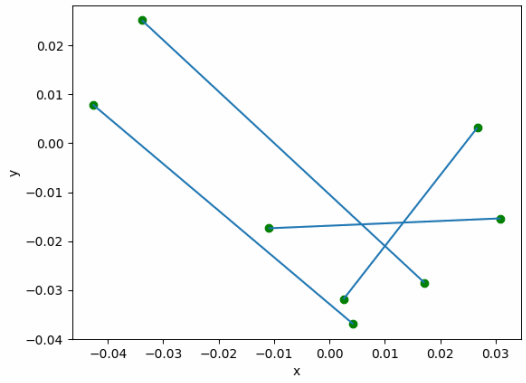
Wykres 14. Zbiór 1.



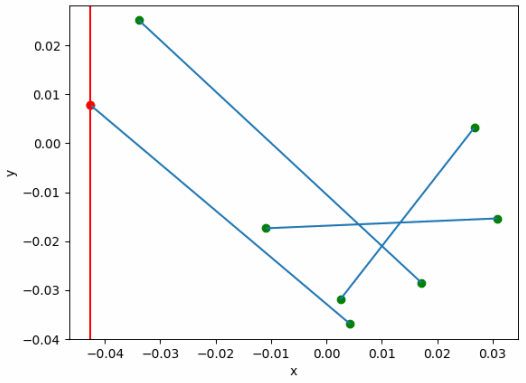
Wykres 15. Zbiór 1.



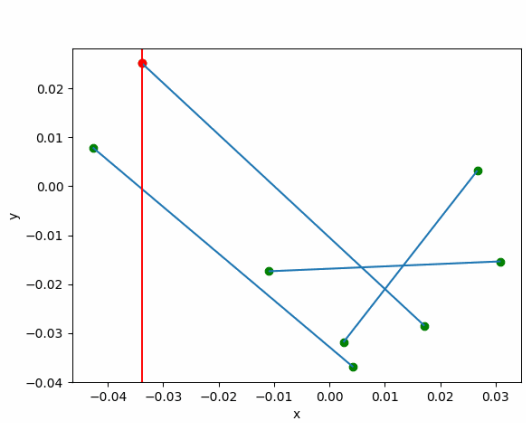
Wykres 16. Zbiór 1.



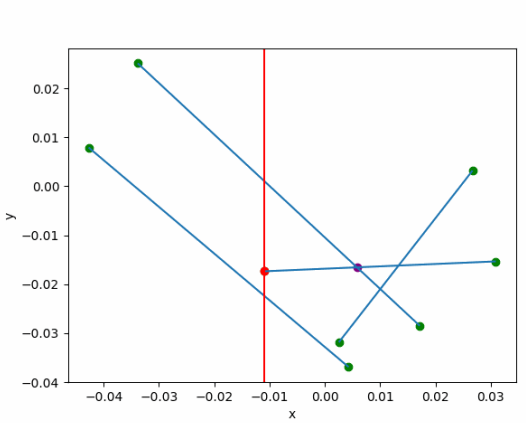
Wykres 13. Zbiór 2.



Wykres 14. Zbiór 2.

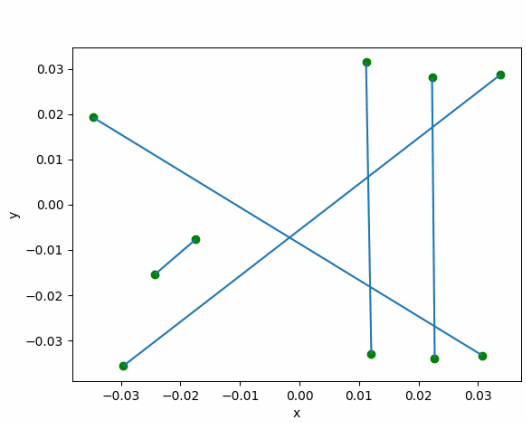


Wykres 15. Zbiór 2.

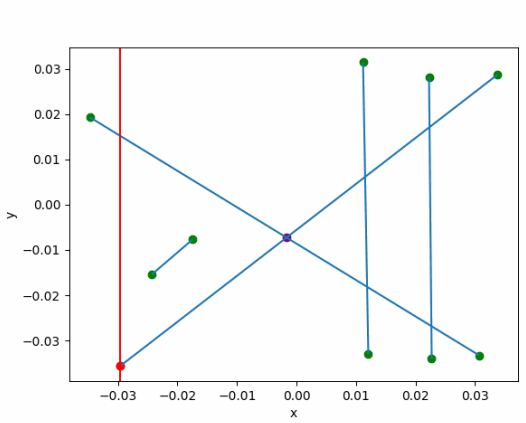


Wykres 16. Zbiór 2.

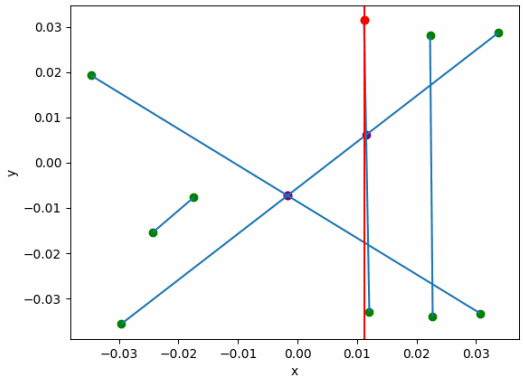
5.2 Algorytm wyznaczający wszystkie przecięcia odcinków



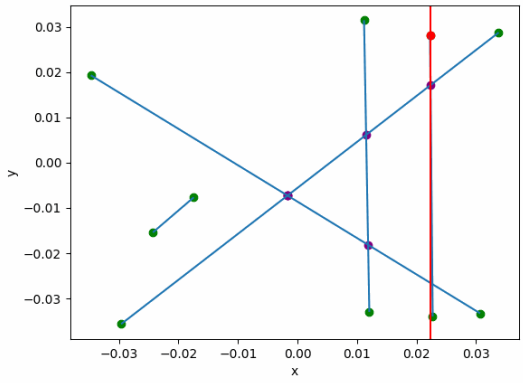
Wykres 13. Zbiór 3.



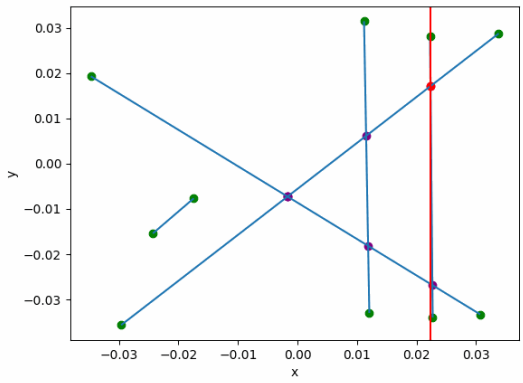
Wykres 14. Zbiór 3.



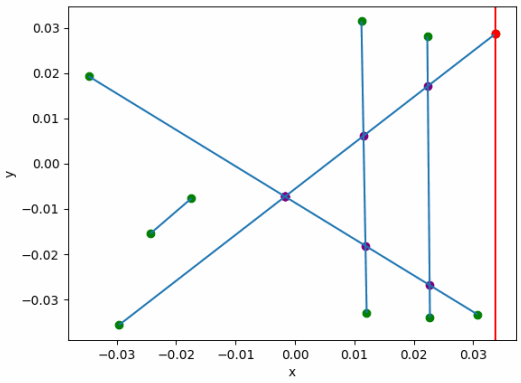
Wykres 7.



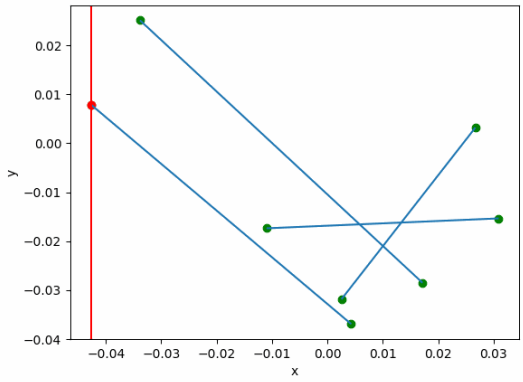
Wykres 8.



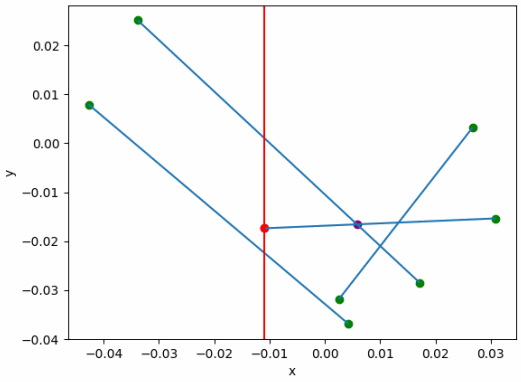
Wykres 9.



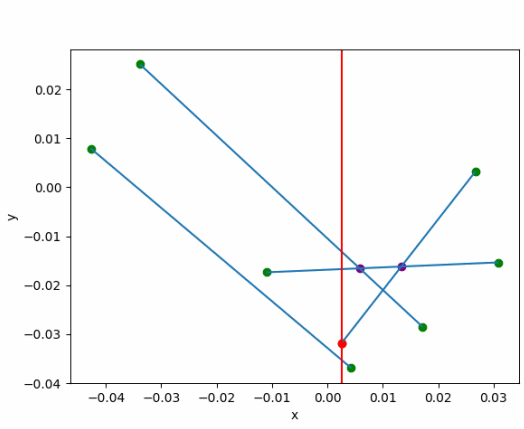
Wykres 10.



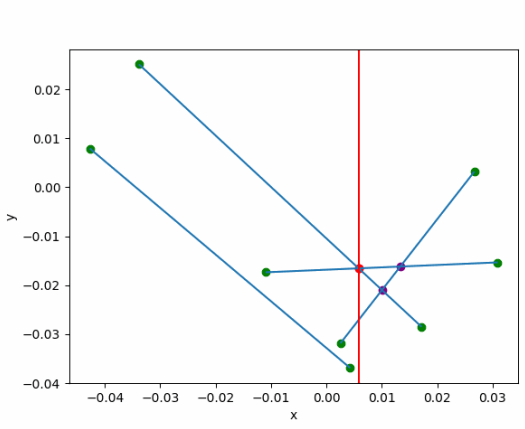
Wykres 11.



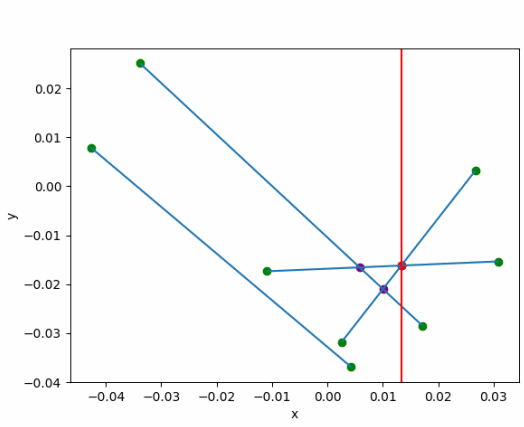
Wykres 12.



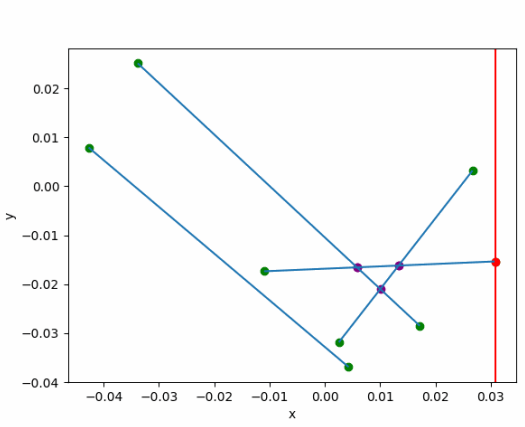
Wykres 7.



Wykres 8.



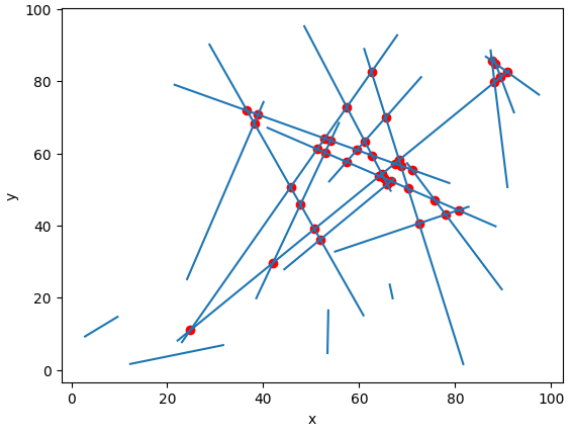
Wykres 9.



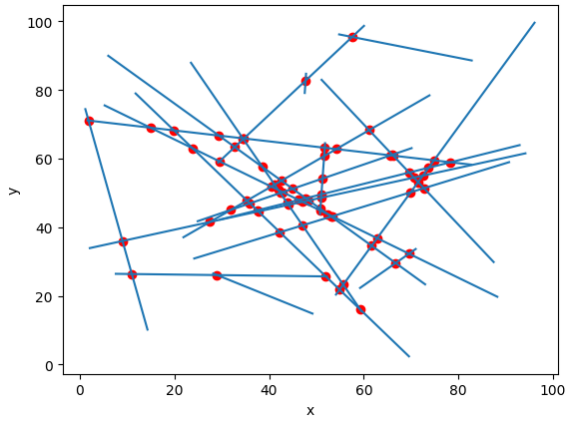
Wykres 10.

6 Przecięcia odcinków dla zbiorów o mocy 20

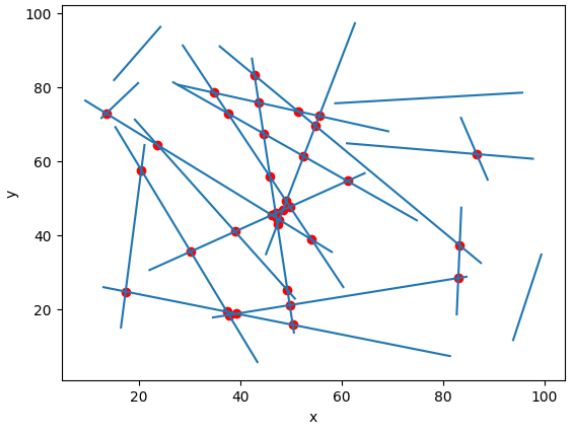
Poniżej znajdują się wyniki drugiego algorytmu dla wygenerowanych zbiorów o nieco większej mocy.



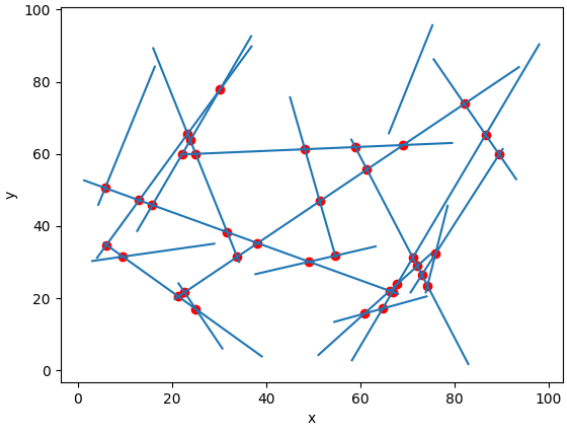
Wykres 7.



Wykres 8.



Wykres 9.



Wykres 10.

7 Wnioski

Algorytmy po przetestowaniu na powyższych zbiorach działają poprawnie oraz umożliwiają ich wizualizację krok po kroku. Nie zauważono żadnych błędów w zbiorach wynikowych.