Akademia Górniczo-Hutnicza
im. Stanisława Staszica
w Krakowie


# Algorytmy Geometryczne

## Laboratorium nr 4

Tomasz Smyda

11 grudnia 2023

# Spis treści

# 1 Wstęp

## 1.1 Opis ćwiczenia

W ćwiczeniu laboratoryjnym nr 4 należało zaimplementować algorytm zamiatania, który stwierdza czy zadane na płaszczyźnie odcinki się przecinają oraz wyznacza punkty ich przecięć. W celu wykonania ćwiczenia należało zaimplementować:

- Funkcję, która w sposób pseudolosowy generuje zadaną ilość odcinków na płaszczyźnie
- Algorytm zamiatania, który stwierdza czy dowolne dwie pary odcinków zadanych na wejściu się przecinają
- Algorytm zamiatania, który dla zadanych odcinków wejściowych wyznacza punkty, które są przecięciami zadanych odcinków.

## 1.2 Biblioteki

Ćwiczenie zostało wykonane przy użyciu narzędzia Jupyter Notebook z wykorzystaniem języka Python w wersji 3.9.18. Do wykonania rysunków oraz narzędzia do zadawania wejściowych wielokątów użyłem biblioteki matplotlib. Do generowania pseudolosowych współrzędnych użyłem biblioteki random. Podczas kolorowania wierzchołków przyjąłem następującą konwencję:

- Punkt początkowy odcinka
- Punkt końcowy odcinka
- Punkt przecięcia odcinków

## 1.3 Specyfikacja

Wszystkie obliczenia były prowadzone na komputerze Lenovo Legion 5 Pro z systemem operacyjnym Windows 11 Home w wersji 22H2, procesorem Intel Core i7-11800H.

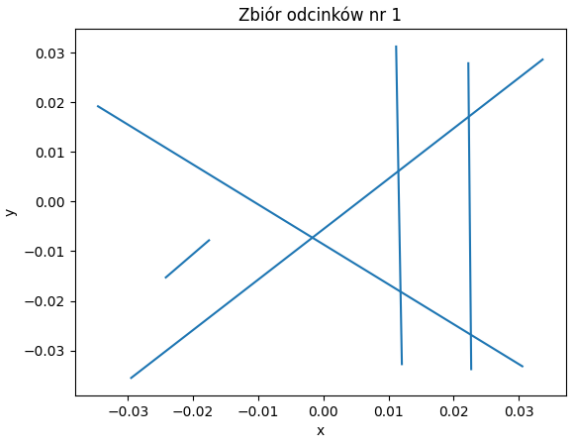
## 1.4 Plan i sposób wykonania ćwiczenia

1. Przygotowuję funkcję generującą zadaną liczbę odcinków o współrzędnych losowych
2. Implementuję algorytm sprawdzający czy dowolne dwa odcinki zadane na wejściu się przecinają
3. Implementuję algorytm wyznaczający zbiór punktów przecięć zadanych odcinków

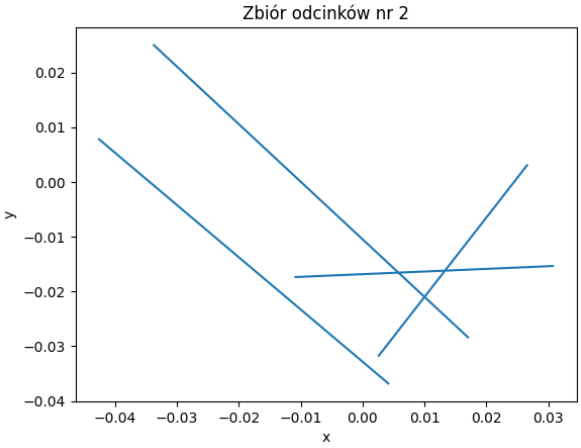
# 2 Realizacja ćwiczenia

## 2.1 Przygotowanie funkcji generującej losowe odcinki oraz zbiorów testowych

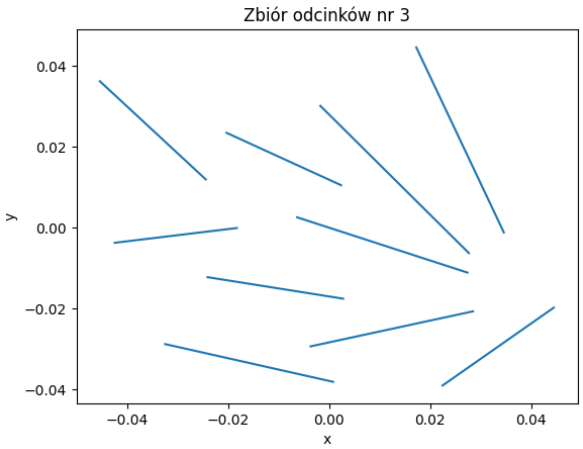
Na poniższych rysunkach przedstawione są zbiory odcinków, których użyłem do sprawdzenia poprawności algorytmów implementowanych w ramach tego ćwiczenia.



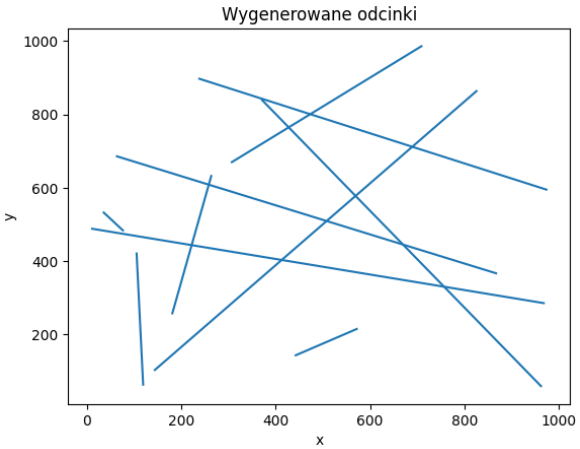

**Rysunek 1:** Zbiór odcinków nr 1




**Rysunek 2:** Zbiór odcinków nr 2




**Rysunek 3:** Zbiór odcinków nr 3




**Rysunek 4:** Losowo wygenerowany zbiór odcinków

## 2.2 Klasa Point oraz Segment

Podczas implementacji algorytmów do reprezentacji punktu oraz odcinka wykorzystałem własnoręcznie zaimplementowane klasy (odpowiednio Point oraz Segment), które posiadają atrybuty oraz metody ułatwiające implementację zadanych algorytmów. Dla przykładu klasa Point zawiera informację na temat współrzędnych punktu oraz informację jak porównywać dwa punkty, natomiast klasa Segment zawiera wartość współczynnika kierunkowego prostej oraz jej wyrazu wolnego, która jest wyznaczona przez ten odcinek, tak aby później łatwo wyznaczyć punkty przecięcia dwóch odcinków oraz stwierdzić, który odcinek znajduje się „wyżej", a który „niżej".

## 2.3 Algorytm zamiatania do wyznaczenia czy odcinki się przecinają

Algorytm na wejściu przyjmuję listę punktów początkowych oraz końcowych dla każdego odcinka, natomiast na wyjściu zwraca wartość logiczną, zależną od tego czy którakolwiek para z zadanych odcinków się przecina.

### 2.3.1 Struktura stanu miotły oraz struktura zdarzeń

Do struktury stanu miotły użyłem SortedSetu z biblioteki sortedcontainers, która przechowuje odcinki posortowane względem drugiej współrzędnej. Jest to struktura reprezentowana przez drzewo czerwono-czarne, co umożliwia operacje wstawiania, usuwania, znajdywania poprzednika/następnika w czasie $O(\log n)$.

Do struktury zdarzeń użyłem zwykłej listy, którą sortuję względem pierwszej współrzędnej punktów po wstawieniu do niej punktów początkowych oraz końcowych odcinków. Mogę to zrobić, ponieważ algorytm zakończy działanie jeżeli znajdzie pierwszą parę przecinających się odcinków, zatem nie potrzebuję operacji wstawiania/usuwania do struktury. Sortowanie ma złożoność $O(n \log n)$, nastomiast wyciąganie każdego elementu jest rzędu $O(1)$, zatem nie tracę na całkowitej złożoności algorytmu.

### 2.3.2 Główny algorytm

Na początku wszystkie punkty początkowe oraz końcowe zadanych odcinków wrzucam do struktury zdarzeń oraz sortuję je po pierwszej współrzędnej. Następnie dla każdego punktu, który wyciągnę ze struktury zdarzeń sprawdzam czy jest on punktem początkowym odcinka czy końcowym. Jeżeli jest początkowym to dodaję go do struktury stanu miotły oraz sprawdzam czy przecina się z którymkolwiek ze swoich sąsiadów jeśli tak to kończę działanie algorytmu z wartością True. Jeżeli punkt jest punktem końcowym odcinka to usuwam ten odcinek ze struktury stanu miotły.

## 2.4 Algorytm do wyznaczenia punktów przecięcia zadanych odcinków

Algorytm na wejściu przyjmuję listę punktów początkowych oraz końcowych dla każdego odcinka, natomiast na wyjściu zwraca listę współrzędnych punktów, będącymi punktami przecięć zadanych odcinków oraz informację, które odcinki wyznaczają dany punkt (w postaci indeksów odcinków).

### 2.4.1 Struktura stanu miotły oraz struktura zdarzeń

O ile struktura stanu miotły nie zmienia się od tej użytej poprzednim razem, to w tym przypadku będziemy dodawać posortowane punkty względem pierwszej współrzędnej do struktury zdarzeń, więc potrzebujemy narzędzia, które umożliwi nam to w jak najkrótszym czasie. Zdecydowałem się na tą samą reprezentację co struktura stanu miotły - SortedSet.

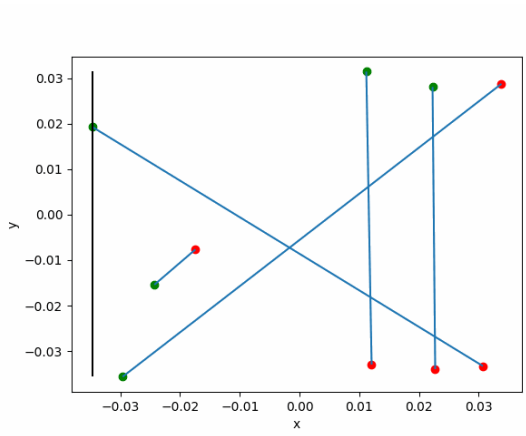
### 2.4.2 Główny algorytm

Początek algorytmu nie różni się od poprzedniego - po kolei wstawiam punkty do struktury zdarzeń. Obsługa dla wyciągniętych punktów początkowych oraz końcowych odcinków różni się tym, że gdy wykryję przecięcie nie kończę działania algorytmu, tylko dodaję wyznaczony punkt przecięcia do struktury zdarzeń. Dla wyciągniętego punktu przecięcia dodaję go oraz indeksy odcinków, które wyznaczają ten punkt do listy, która przechowuje mi punkty przecięcia,

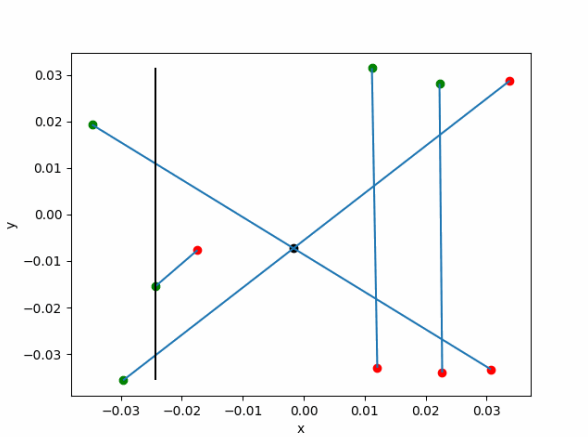
następnie zamieniam kolejność odcinków w strukturze stanu miotły oraz sprawdzam czy istnieje przecięcie każdego z tych odcinków z nowymi sąsiadami. Na koniec zwracam listę przechowującą mi wyznaczone punkty przecięcia.

## 2.5 Wizualizacja dla zadanych danych testowych

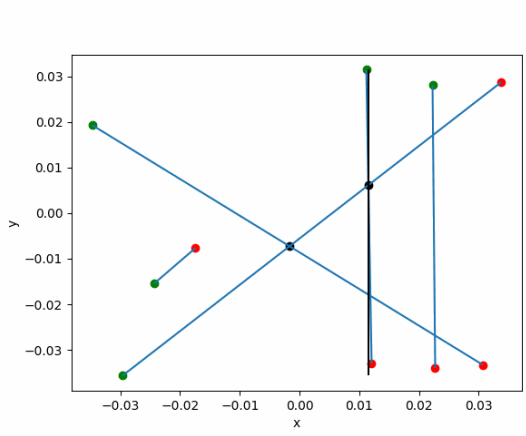
### 2.5.1 Zbiór odcinków nr 1



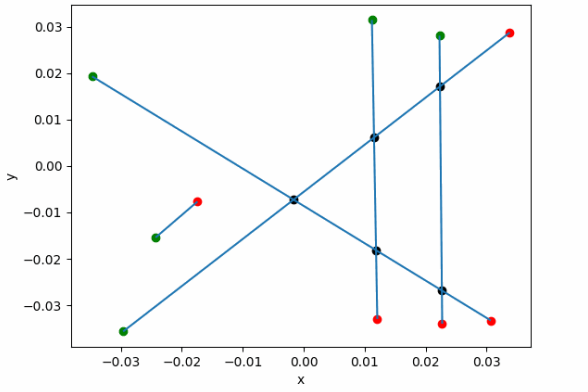

**Rysunek 5:** Stan początkowy algorytmu




**Rysunek 6:** Stan "szukania" przecięć

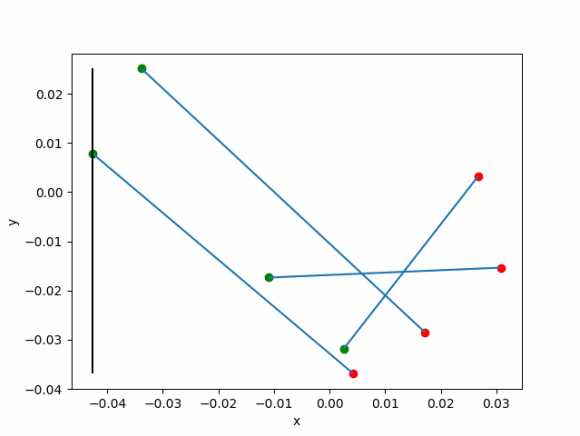



**Rysunek 7:** Stan "szukania" przecięć

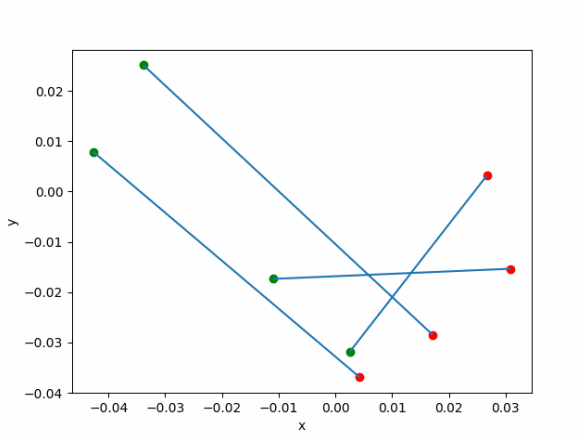



**Rysunek 8:** Wizualizacja znalezionych punktów przecięć

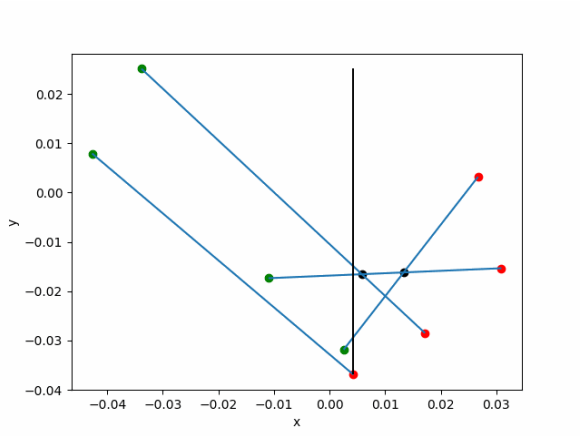
### 2.5.2 Zbiór odcinków nr 2



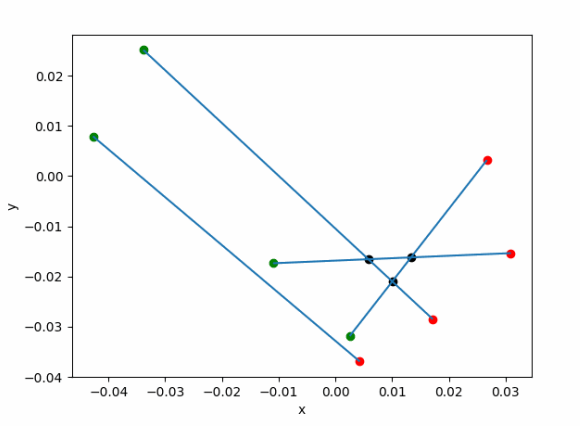

**Rysunek 9:** Stan początkowy algorytmu




**Rysunek 10:** Stan "szukania" przecięć

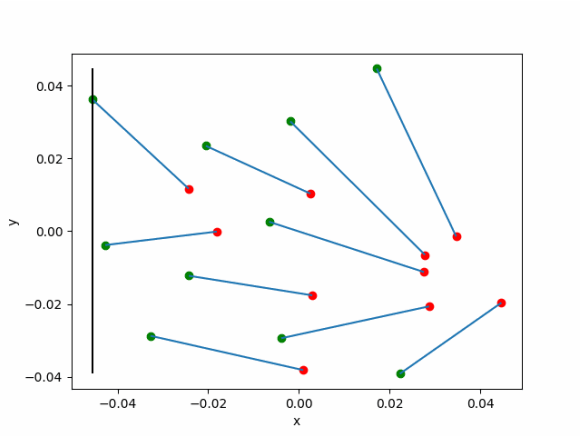



**Rysunek 11:** Stan "szukania" przecięć

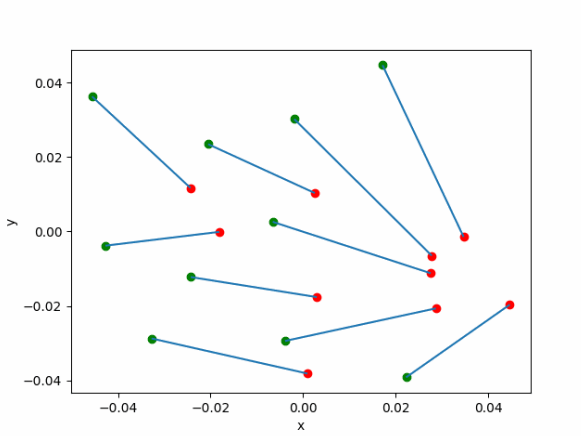



**Rysunek 12:** Wizualizacja znalezionych punktów przecięć

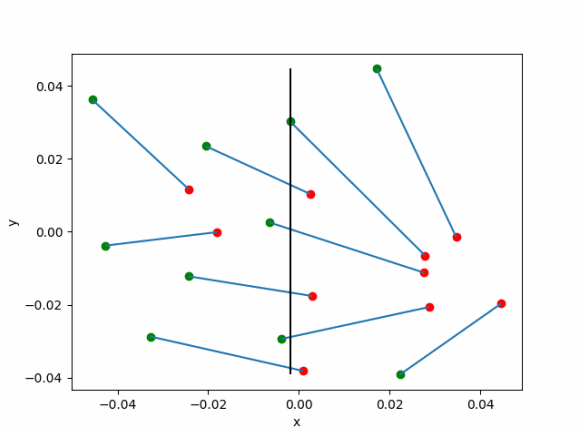
### 2.5.3 Zbiór odcinków nr 3



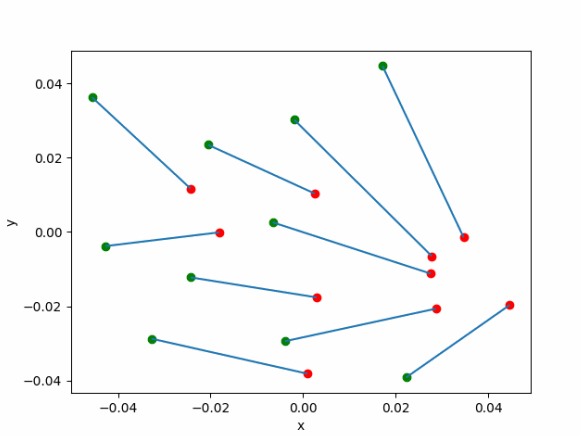

**Rysunek 13:** Stan początkowy algorytmu




**Rysunek 14:** Stan "szukania" przecięć

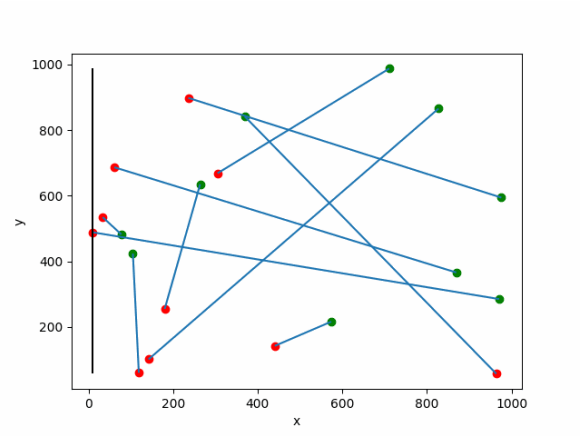



**Rysunek 15:** Stan "szukania" przecięć

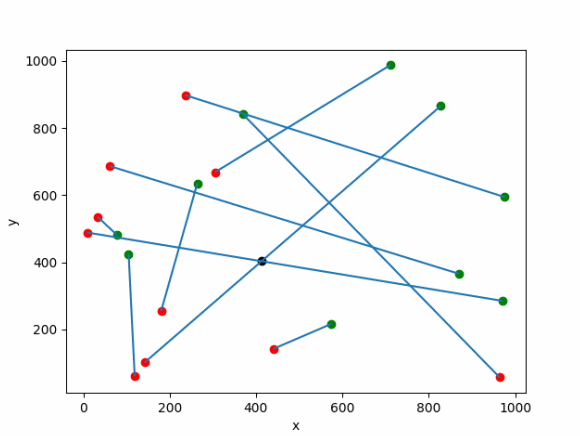



**Rysunek 16:** Wizualizacja znalezionych punktów przecięć

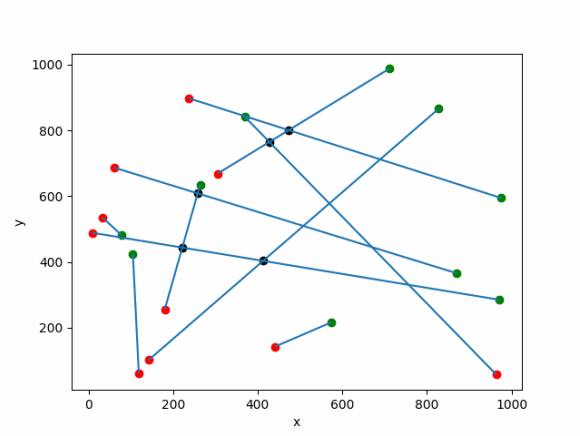
### 2.5.4 Zbiór odcinków nr 4



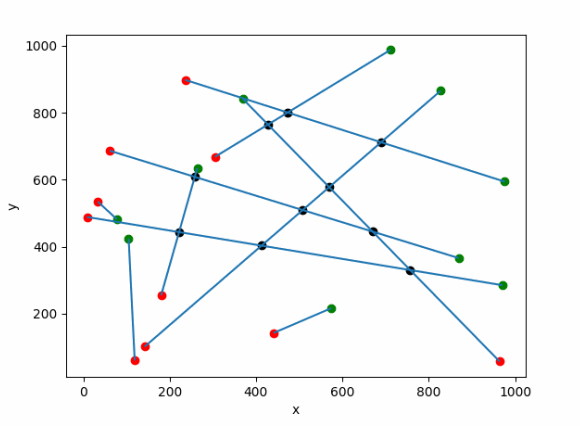

**Rysunek 17:** Stan początkowy algorytmu




**Rysunek 18:** Stan "szukania" przecięć




**Rysunek 19:** Stan "szukania" przecięć




**Rysunek 20:** Wizualizacja znalezionych punktów przecięć

# 3 Wnioski

Na przedstawionych rysunkach będących wizualizacjami działania algorytmu widzimy, że zwracają one poprawne wyniki dla podanych zbiorów testowych. Biorąc pod uwagę fakt, że zbiory te uwzględniają przypadki skrajne, możemy stwierdzić, że algorytmy zostały poprawnie zaimplementowane, a ich rezultaty pokrywają się z oczekiwaniami.