# Juliusz Wasieleski Informatyka II rok, grupa 1

Algorytmy geometryczne, laboratorium 4 - sprawozdanie

# 1.Opis ćwiczenia

W zadaniu numer 4 należało zaimplementować algorytm zamiatania, który wyznacza punkty przecięcia odcinków na płaszczyźnie. Należało zaimplementować:

* funkcję umożliwiającą generowanie w losowy sposób zadanej ilości odcinków we współrzędnych 2D,
* algorytm zamiatania sprawdzający, czy dowolne dwie pary odcinków w zadanym zbiorze przecinają się,
* algorytm wyznaczający wszystkie przecięcia odcinków w zbiorze,
* zwrócenie liczby wszystkich punktów przecięcia, ich współrzędne oraz odcinki, które się przecinają.

# 2. Środowisko, biblioteki, założenia oraz użyte narzędzia

Ćwiczenie wykonałem w Jupyter Notebook i napisałem w języku Python. Podczas wykonywania zadania korzystałem z bibliotek *pandas* oraz *numpy.*

Do rysowania wykresów użyłem narzędzia graficznego z laboratorium, które jest oparte o bibliotekę *matplotlib*.

Podczas kolorowania wierzchołków przyjąłem konwencję z wykładu:

* **Niebieski** – początki i końce odcinków,
* **Czerwony** – punkty przecięcia

Wszystkie obliczenia prowadziłem na komputerze Lenovo Y50-70 z systemem Windows 10 Pro w wersji 10.0.19045, procesor Intel Core i7-4720HQ 2.60GHz, 2601 MHz, rdzenie: 4, procesory logiczne: 8.

# 3. Plan i sposób wykonania ćwiczenia

1. Przygotowuję funkcje do wczytywania punktów zaznaczonych myszką, zapisu ich do pliku   
   i wczytywania punktów z pliku oraz generowania losowych punktów.
2. Implementuję lub wybieram z gotowych struktury: zdarzeń oraz stanu
3. Implementuję algorytm zamiatania

# 4. Wykonanie ćwiczenia

## 4.1 Struktura stanu miotły oraz struktura zdarzeń

W pierwszej kolejności zaimplementowałem funkcje zapisujące wprowadzone punkty do pliku oraz odczytujące je z pliku (*save\_polygon\_to\_file*). Wtedy też zaimplementowałem funkcje do generowania losowych zbiorów linii.

## 4.2 Struktura stanu miotły oraz struktura zdarzeń

Punkty oraz odcinki są reprezentowane odpowiednio klasami *Point* oraz *Section*. Każdy punkt ma pola będące jego współrzędnymi x oraz y zaokrąglonymi do 10 miejsca po przecinku, aby uniknąć błędów podczas porównywania ich współrzędnych oraz pole index - w programie używane do określenia, do jakiego odcinka w tablicy należy wierzchołek. Wartość None implikuje, że wierzchołek jest przecięciem dwóch wierzchołków i nie należy do jednego wierzchołka.

Pierwszym algorytmem, który należało zaimplementować jest algorytm sprawdzający, czy dowolne dwie pary odcinków w zadanym zbiorze przecinają się. Implementacja została umieszczona w metodach klasy *Section: check\_intersection* oraz *find\_intersection\_point*.

Użyłem *SortedSetu* jako struktury stanu miotły, która przechowuje posortowane odcinki względem współrzędnej y , a w Pythonie występuje w bibliotece *sortedcontainers*. Klasa ta jest reprezentowana przez drzewo czerwono-czarne, dzięki czemu możliwe jest wstawianie, usuwanie oraz znajdowanie poprzednika/następnika elementu w czasie O(logn). Do struktury zdarzeń użyłem ręcznie zaimplementowanego drzewa BST, w którym kluczami są współrzędne x-owe punktów. Dodatkowo polami w tej klasie są: obiekt klasy punkt, który jest węzłem w tym drzewie, oraz pola trzymające wskaźniki na prawe, lewe dziecko oraz parenta.

## 4.3 Algorytm zamiatania (znajdujący przecięcia)

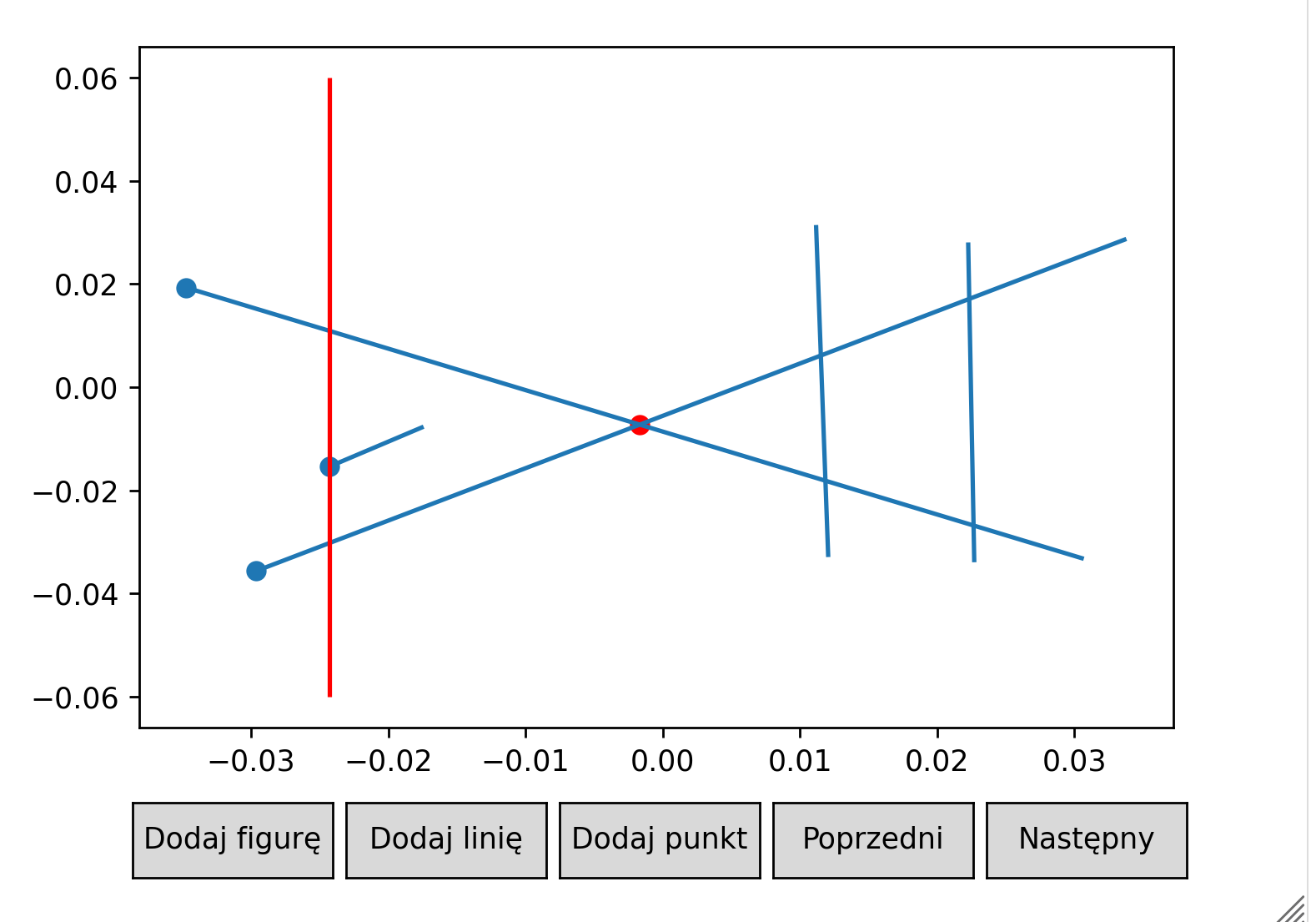
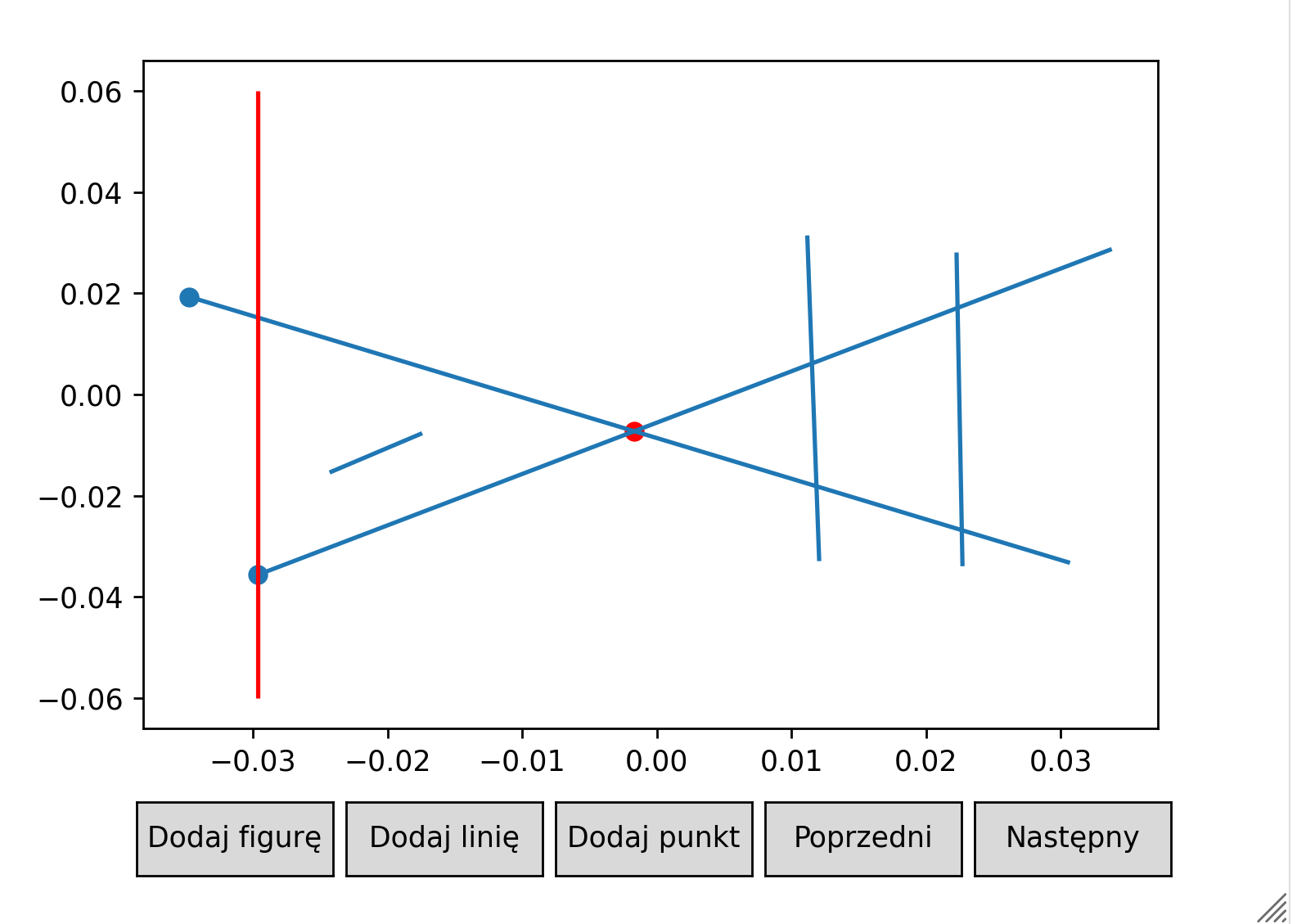
Algorytm szuka punktu przecięcia w oparciu o algorytm zamiatania przedstawiony na wykładzie, ale jego struktura wygląda trochę inaczej ze względu na warunki w if-ach (nie miało sensu rozbijać osobno struktury stanu oraz struktury zdarzeń).

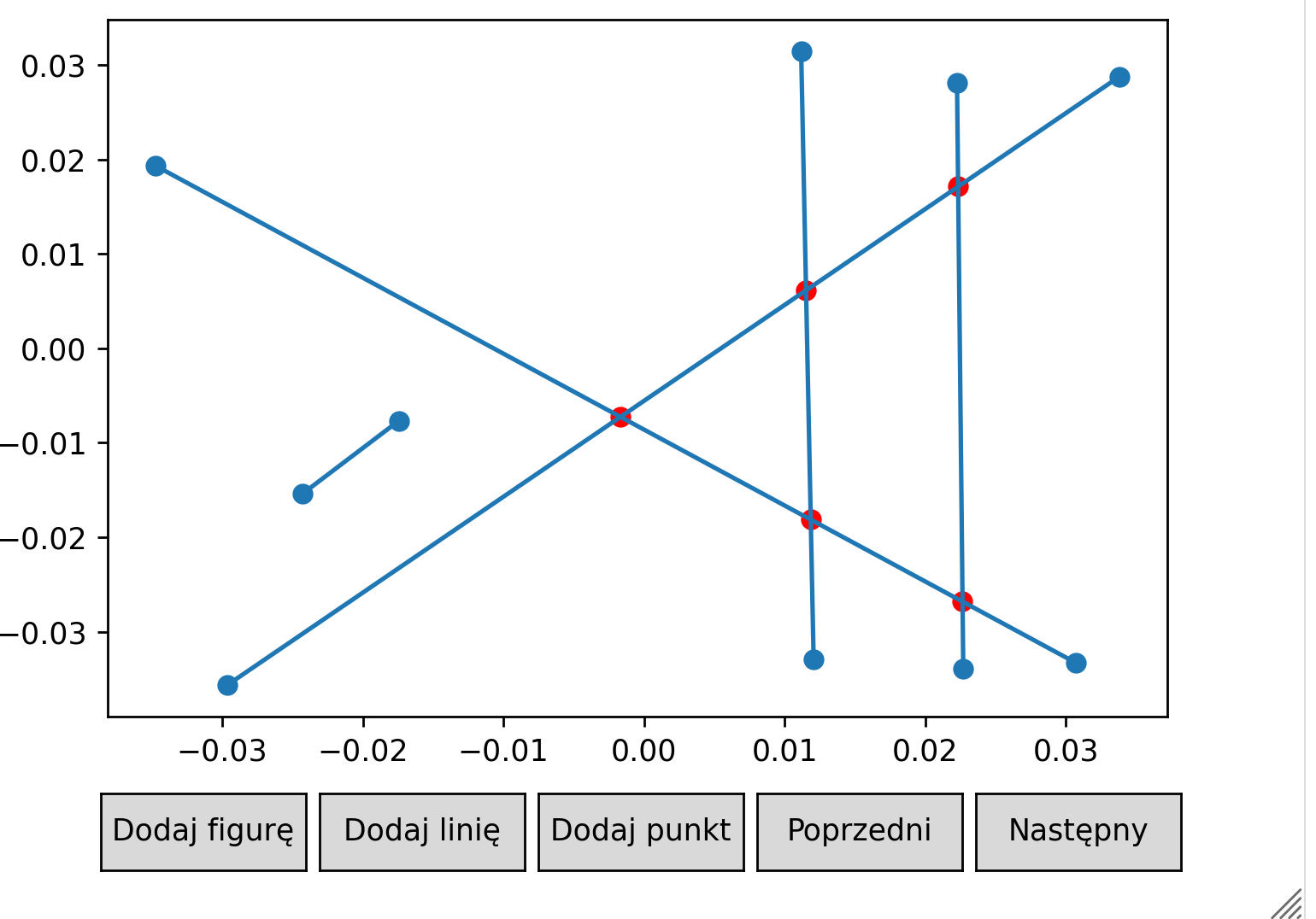
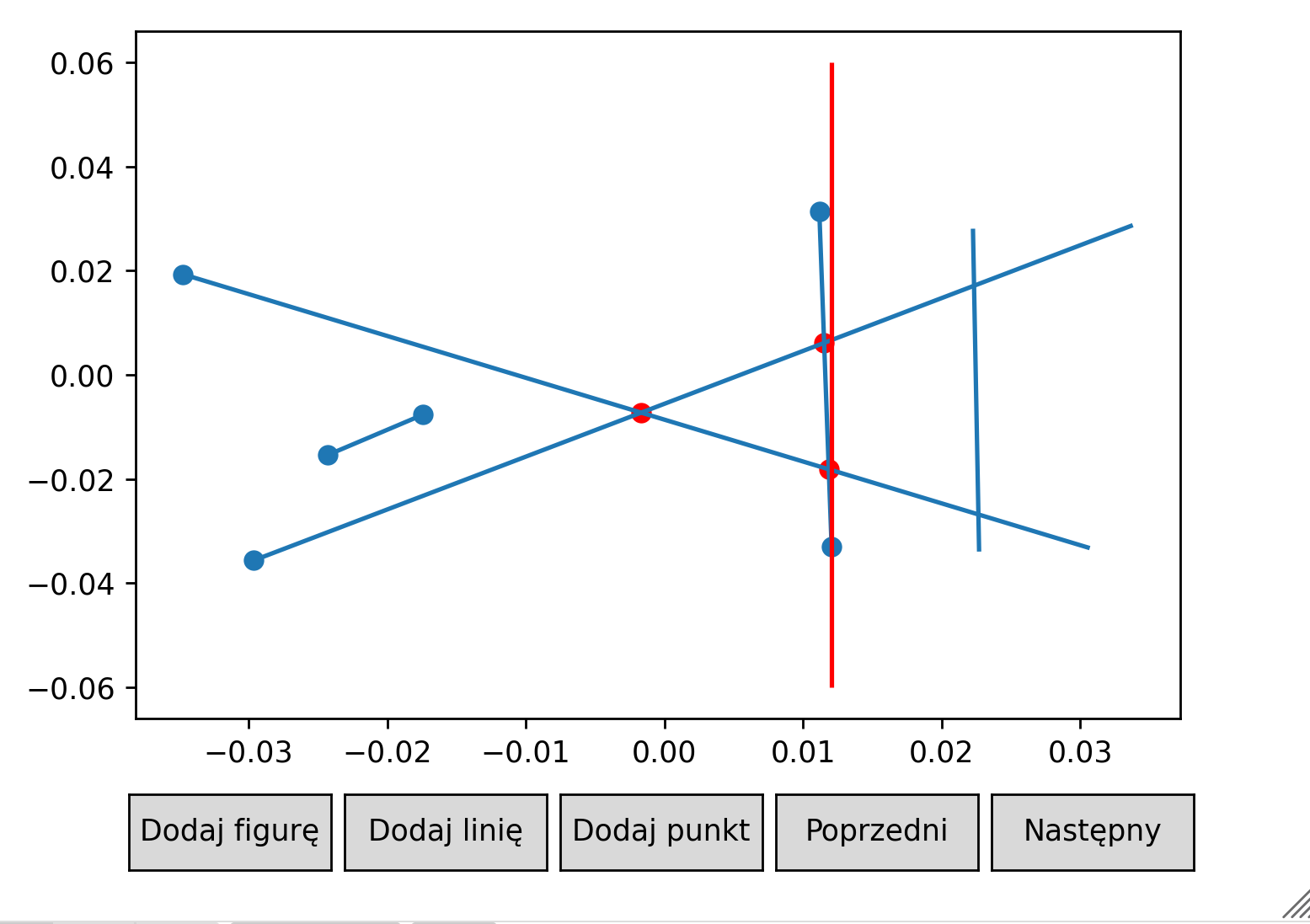
Przechodzę przez wszystkie zdarzenia ze struktury zdarzeń (chodzę po kolejnych punktach będących Node-ami w drzewie BST). Jeśli zdarzenie należy więcej niż do jednej linii, to znaczy, że mamy do czynienia z punktem przecięcia. Wtedy zamieniamy ze sobą porządek linii które się przecinają   
w strukturze stanu (ponieważ za punktem przecięcia ta, która była niżej będzie wyżej i na odwrót). Żeby udało się to dobrze zrobić przesuwam współrzędne o epsilon dla punktów (EPS\_POINT = 1e-5). Następnie jeśli odcinek, który za punktem przecięcia będzie wyżej, nie jest najwyższym odcinkiem   
w strukturze stanu, to szukam jego przecięcia z odcinkiem o 1 wyżej. Takie przecięcie dodaję do struktury zdarzeń. Dodawanie jest realizowane przez metodę *insert* zaimplementowaną w klasie drzewa BST. Jeśli metoda ta zwróci fałsz oznacza to, że nie udało się wstawić tego przecięcia ponieważ istnieje już ono w naszym drzewie, w ten sposób nie dodaję kilkukrotnie tego samego punktu. Analogicznie sprawdzam przecięcie prostej, która za przecięciem znajduje się niżej. O ile nie jest ona najniższą w strukturze stanu, to szukam jej przecięcia i dodaję je do struktury zdarzeń.

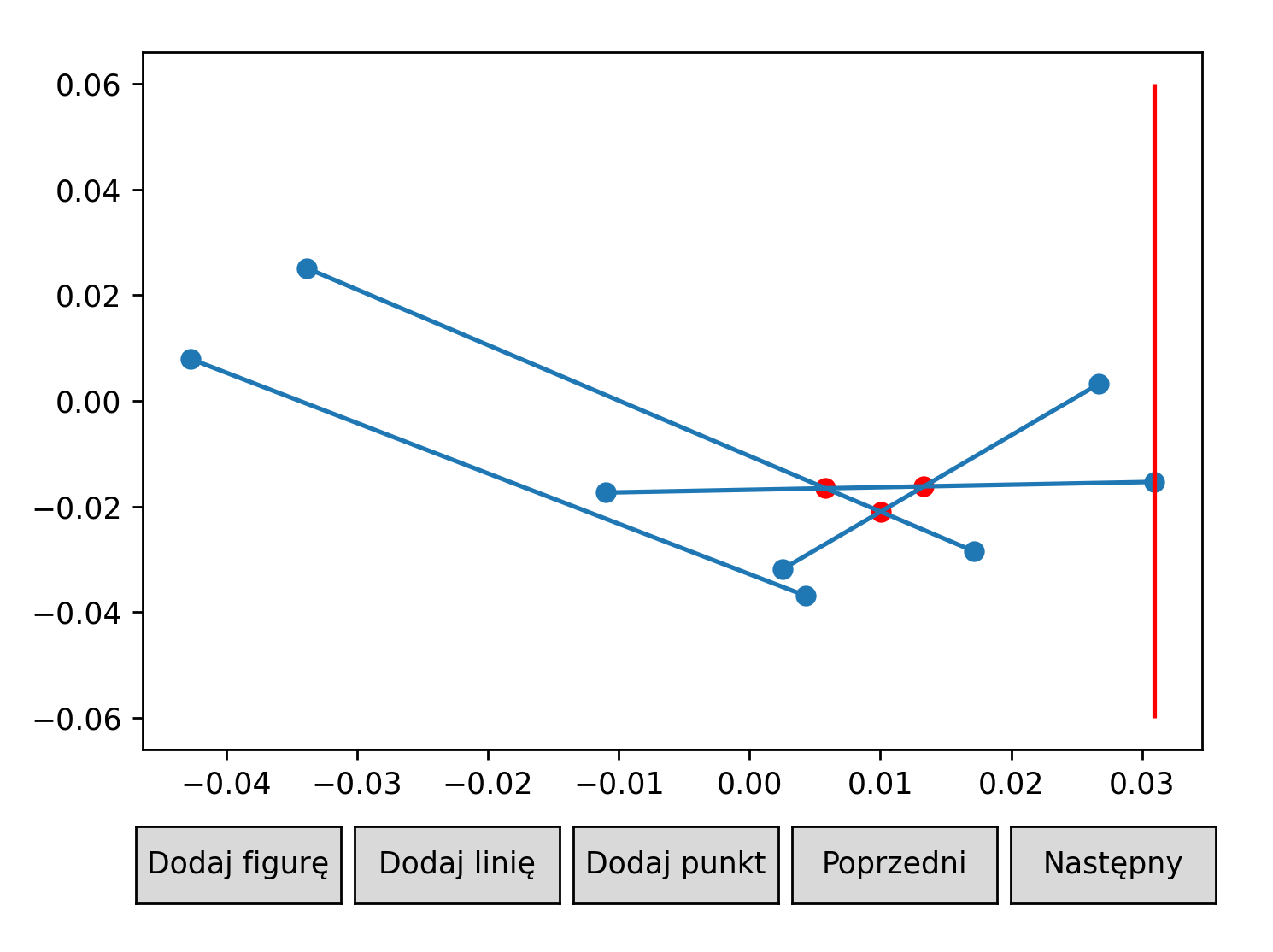
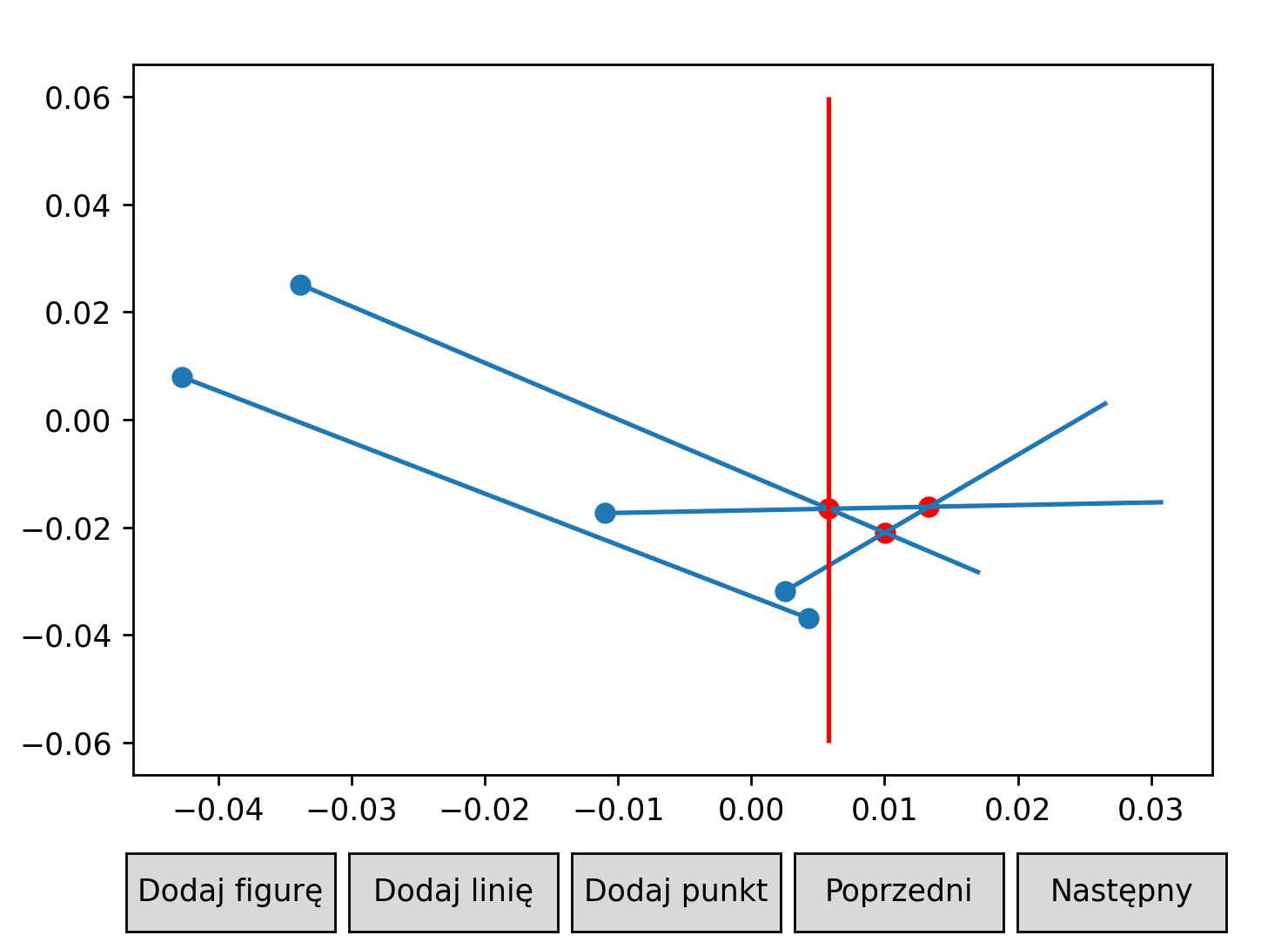
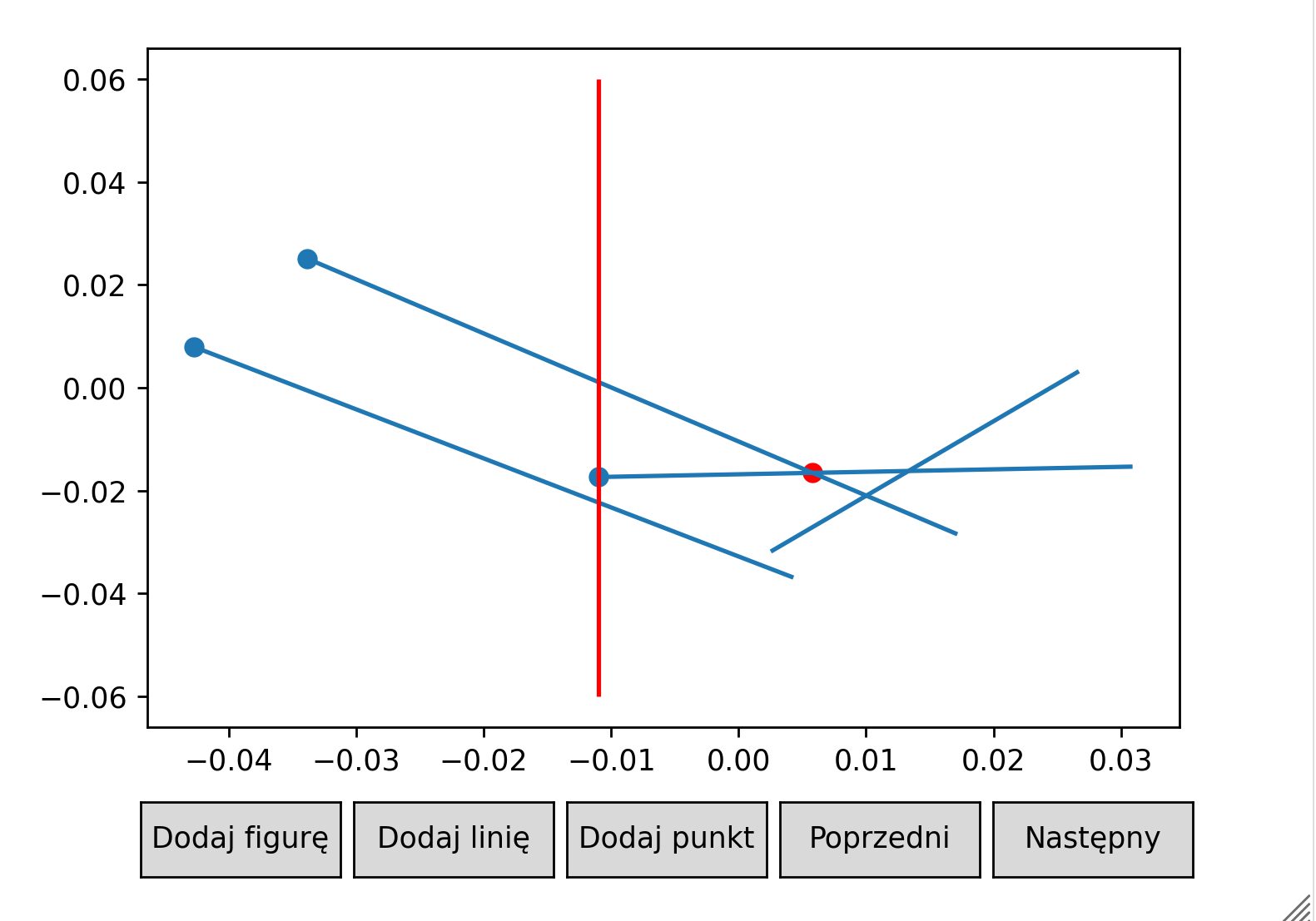
Jeśli punkt wyciągnięty ze struktury zdarzeń jest tylko na jednej linii to znaczy, że jest początkiem lub końcem któregoś z odcinków. Jeśli mamy do czynienia z lewym końcem odcinka dodajemy ten odcinek do struktury stanu, a następnie szukamy przecięcia z linią powyżej i poniżej, analogicznie jak opisałem to w poprzednim akapicie. Jeśli natomiast punkt wyciągnięty ze struktury zdarzeń jest prawym końcem, to sprawdzam czy po prawej stronie od miotły ten odcinek, którego prawy koniec rozważam ma jakieś przecięcia. Jeśli tak, to dodaję je do struktury zdarzeń, a na koniec usuwam ten odcinek ze struktury stanu.

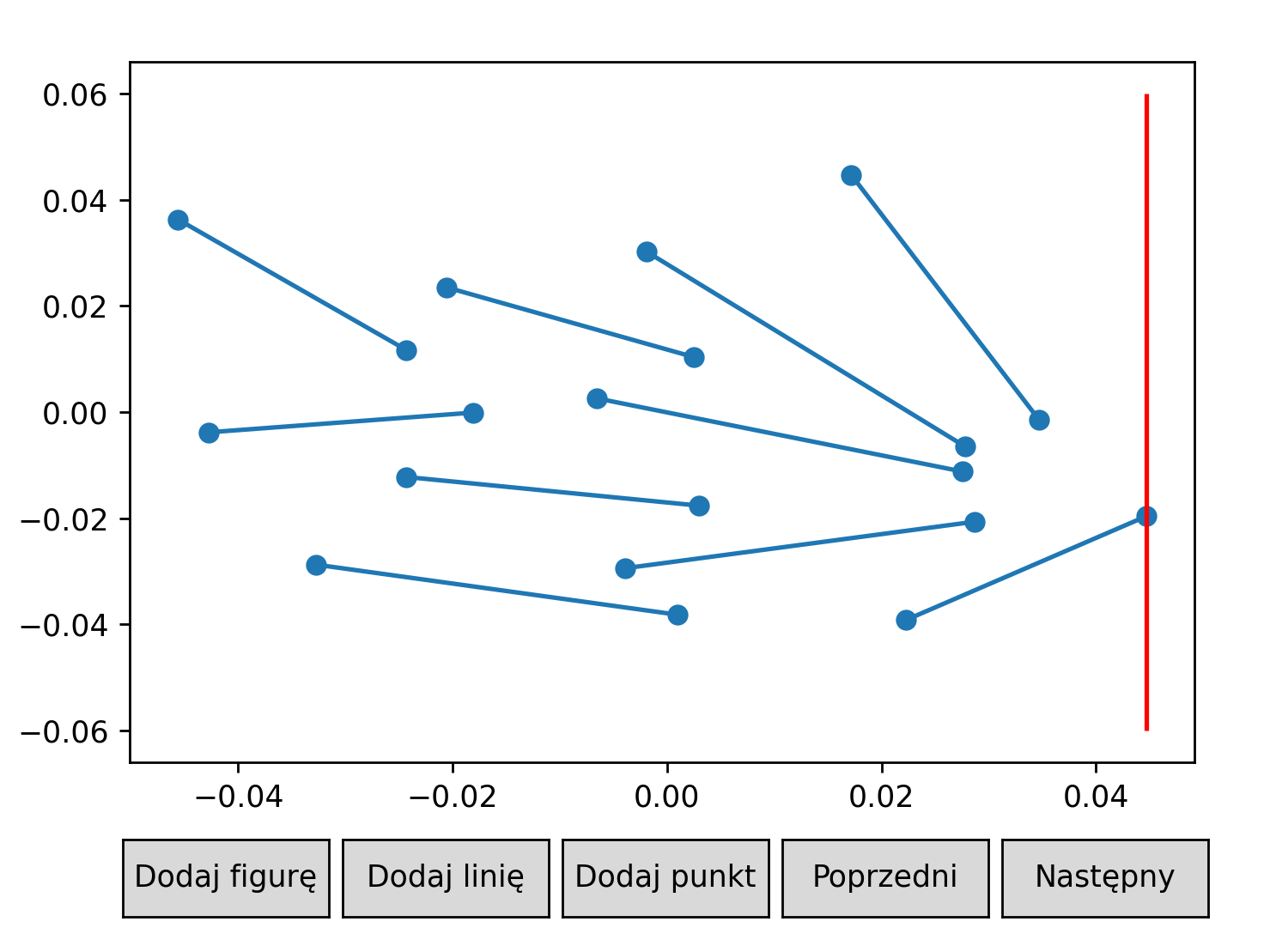
Dodanie przecięcia w wizualizacji odbywa się wtedy, kiedy miotła dojdzie początku drugiego odcinka który się przecina.

Poniżej zamieszam przykładowe momenty działania tego algorytmu dla różnych przykładowych zbiorów linii z folderu tests:









# **5**. Wnioski

Algorytmy po ich przetestowaniu na powyższych zbiorach działają poprawnie. Pamiętając o tym, że w wielu wypadkach są to przypadki skrajne, można stwierdzić, że algorytmy są zaimplementowane poprawnie i działają zgodnie z oczekiwaniami.