Opracowanie wyników laboratoriów

Środowisko

Wszystkie operacje wykonane zostały na komputerze stacjonarnym z procesorem i5-7600k, RAM 32GB. Językiem z jakiego korzystałem był Python w wersji 3.10. Program napisany został w *Jupyter Notebook*.

Sprawdzanie y-monotoniczności

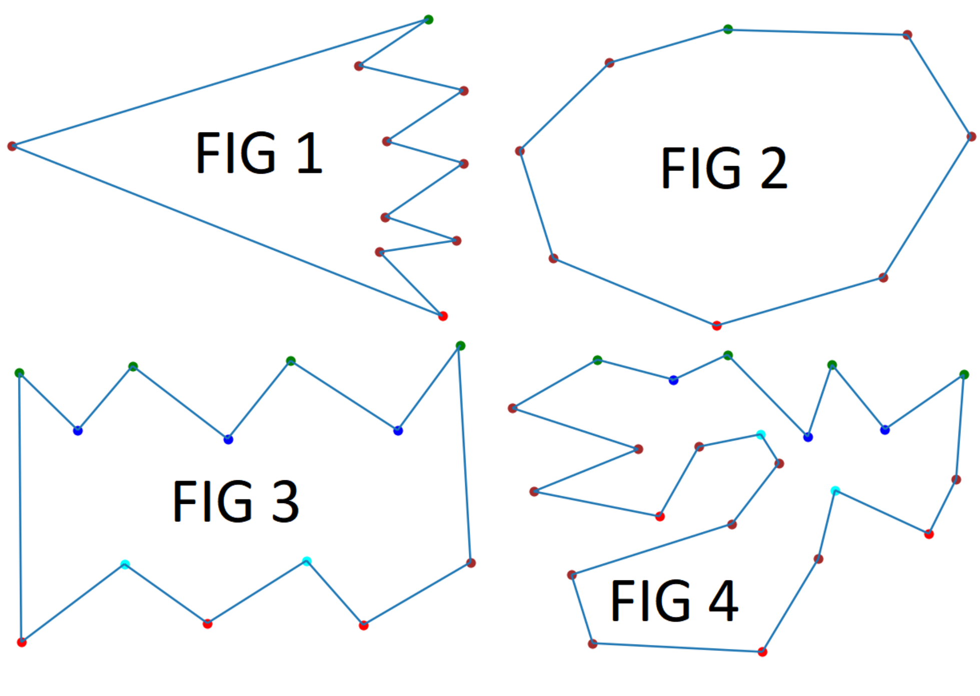
Dodawanie wielokątów odbywa się za pomocą zaproponowanego narzędzia graficznego. Aby określić, czy wielokąt jest y-monotoniczny, wystarczy dla każdego wierzchołka sprawdzić, czy jest on łączący lub dzielący, jeżeli tak, to wielokąt nie jest monotoniczny. Zaimplementowałem to korzystając z mojego wyznacznika **2x2**, a następnie przechodząc po wszystkich wierzchołkach sprawdziłem czy:

$$\exists_{v^i \in V} : v_y^{i-1} > v_y^i \wedge v_y^{i+1} > v_y^i \wedge \det\bigl(v^{i-1}, v, v^{i+1}\bigr) < 0 \ - \ \text{łączący},$$
$$\exists_{v^i \in V} : v_y^{i-1} < v_y^i \wedge v_y^{i+1} < v_y^i \wedge \det\bigl(v^{i-1}, v, v^{i+1}\bigr) < 0 \ - \ \text{dzielący},$$

gdzie v^i oznacza i-ty wierzchołek, $v^{i\pm1}$ - wierzchołki połączone z v^i krawędzią, natomiast v_y oznacza y-ową współrzędną wierzchołka v. Rolą wyznacznika jest określenie wklęsłości kąta.

Podział wierzchołków wielokąta

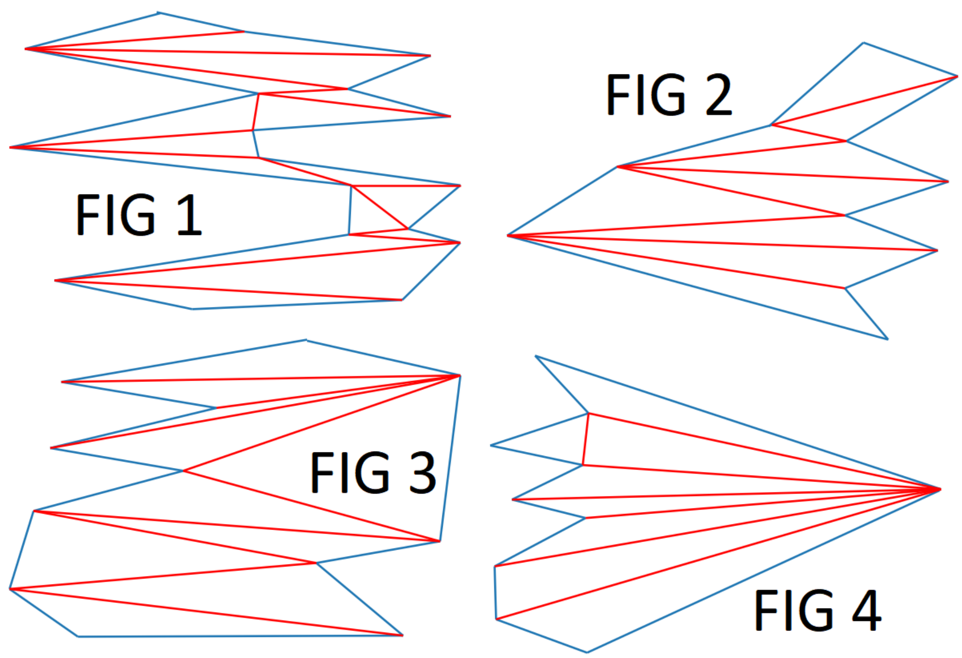
Zgodnie z treścią zadania 3), możemy dokonać podziału odpowiednio na wierzchołki początkowe•, końcowe•, łączące•, dzielące• i prawidłowe•. Do dokonania podziału skorzystałem z wyznacznika **2x2** z punktu wyżej. Poniżej w *Wizualizacji 1* zamieszczam podział na odpowiednie punkty 4 wybranych figur.



Wizualizacja 1: Podział wielokątów.

Triangulacja wielokąta monotonicznego

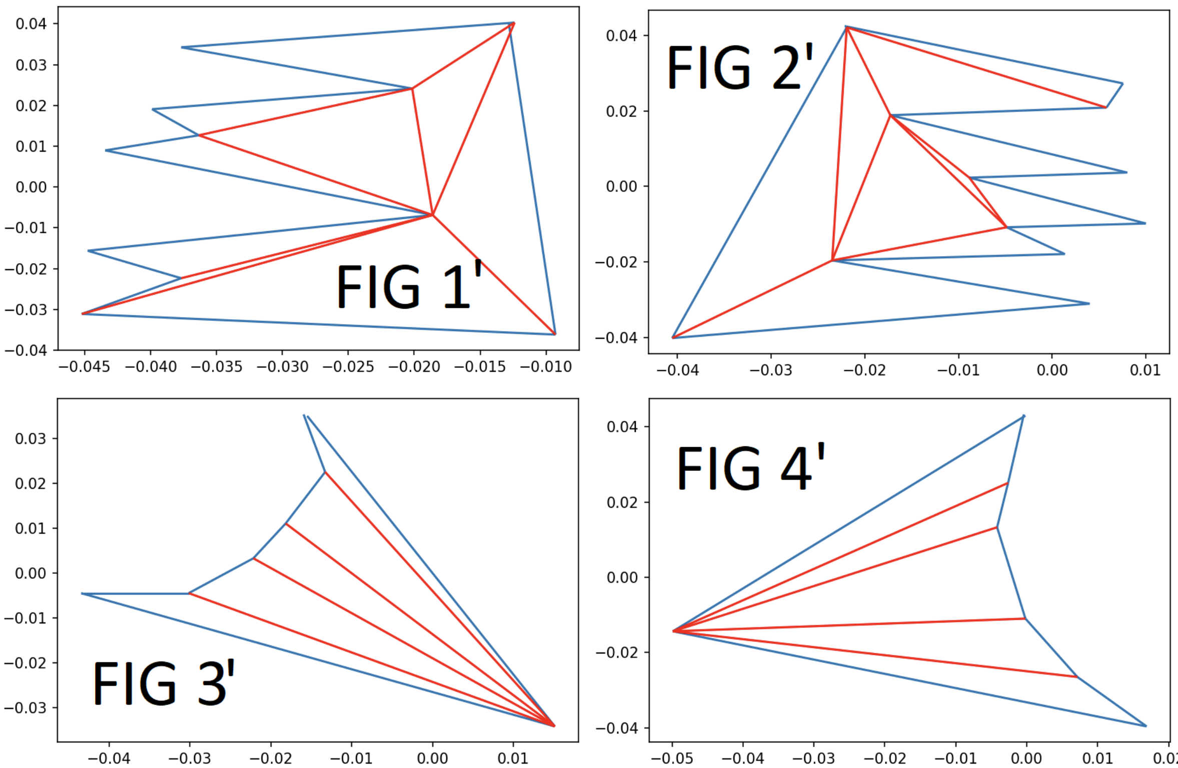
Korzystając z opisanego na wykładzie algorytmu zaimplementowałem funkcje zwracającą listę krawędzi, które pozwalają podzielić **figure** na trójkąty, **triangulateMonotonic(figure)**. Wielokąt przechowuje w postaci listy zawierającej krotki (*idx*, *point*, *class*). *idx* odpowiada za indeks wierzchołka w stworzonym na początku wielokącie. *point* jest dwuelementową listą zawierającą odpowiednio współrzędną x-ową i y-ową punktu. *class* przyjmuje wartości 1 dla punktów należących do lewego łańcucha oraz -1 dla prawego łańcucha. Pozwala to zachować wszystkie informacje o wielokącie po posortowaniu ich malejąco po współrzędnej y-owej, co jest wymagane do dokonania triangulacji. Następnie przeprowadzone są kroki opisane na wykładzie. Metoda zwracająca jedynie dodawane krawędzie co pozwala na zminimalizowanie zajmowanej pamięci. Analogicznie zaimplementowałem funkcje **visualizeTrinagulation(figure)**, która zwracaj listę scen umożliwiających wizualizacje kolejnych kroków. Ponadto możemy wykorzystać funkcje z narzędzia graficznego, aby zapisać utworzony przez nas wielokąt. Poniżej w *Wizualizacji 2* zamieszczam triangulacje 4 wybranych figur monotonicznych.



Wizualizacja 2: Triangulacja wielokątów.

Figury *2* oraz *4* wybrałem jako podobne do siebie, jednak odbite względem osi Y, aby sprawdzić poprawność procedury sprawdzającej, czy trójkąt zawiera się w wielokącie. Figury *1* oraz *3* wybrałem jako sprawdzające poprawne działanie algorytmu dla bardziej skomplikowanych wielokątów. Wielokąty, które wybrałem według mnie dostatecznie sprawdzają działanie algorytmu triangulacji wielokąta monotonicznego.

Poniżej w *Wizualizacji 3* zamieszczam, także figury zadane na laboratoriach.



Wizualizacja 3: Triangulacja wielokątów z laboratoriów.

Wnioski

Jak widzimy, poprawne zaimplementowanie algorytmu triangulacji wielokąta monotonicznego, przy użyciu kopca pozwala na znalezienie poprawnego rozkładu na trójkąty w czasie O(n). W zależności od wykorzystania algorytmu, łatwo można zmienić typ zwracanych danych na listę zawierającą trójkąty. Jednak w przypadku samej wizualizacji działania algorytmu lista dodanych krawędzi wydaje się bardziej słuszna.