In [114…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from bitalg.tests.test3 import Test
from bitalg.visualizer.main import Visualizer
```

Triangulacja wielokątów monotonicznych - problem monitorowania galerii

Przydatne funkcje

In [115…
```
polygon_example_1 = [(5,5), (3,4), (6,3), (4,2), (6,0), (7,1), (8,4)]
polygon_example_2 = [(4, 1), (2, 2), (3, 5), (4, 7), (6, 6), (3, 9), (6,
polygon_example_colors = [4,4,4,4,3,0,2,0,4,4,1]
polygon_example_tri = [(polygon_example_1[0], polygon_example_1[2]),
                       (polygon_example_1[2], polygon_example_1[5]),
                       (polygon_example_1[2], polygon_example_1[6]),
                       (polygon_example_1[6], polygon_example_1[3]),
                       (polygon_example_1[2], polygon_example_1[4]),]
```

In [116…
```
def draw_polygon(polygon):
    vis = Visualizer()
    points = polygon
    vis.add_polygon(polygon, fill=False)
    vis.show()
```

In [117…
```
def draw_polygon_colors(polygon,colors):
    points_start=[]
    points_end=[]
    points_connect=[]
    points_divide=[]
    points_regular=[]
    for i in range(len(polygon)):
        if colors[i]==0:
            points_start.append(polygon[i])
        elif colors[i]==1:
            points_end.append(polygon[i])
        elif colors[i]==2:
            points_connect.append(polygon[i])
        elif colors[i]==3:
            points_divide.append(polygon[i])
        elif colors[i]==4:
            points_regular.append(polygon[i])

    vis = Visualizer()
    colors_start = ['green']
    color_end=['red']
    color_connect=['blue']
    color_divide=['cyan']
```

```
    color_regular=['#3B240B']
    vis.add_polygon(polygon, fill=False)
    vis.add_point(points_start, color=colors_start)
    vis.add_point(points_end, color=color_end)
    vis.add_point(points_connect, color=color_connect)
    vis.add_point(points_divide, color=color_divide)
    vis.add_point(points_regular, color=color_regular)
    vis.show()
```

In [118…

```
def draw_polygon_tri(polygon,tri):
    plt.close()
    vis = Visualizer()
    points = polygon
    tri_line_segments = tri
    vis.add_polygon(points, fill=False)
    vis.add_point(points)
    vis.add_line_segment(tri_line_segments, color='red')
    vis.show()
```

Wprowadzenie

Celem ćwiczenia jest:

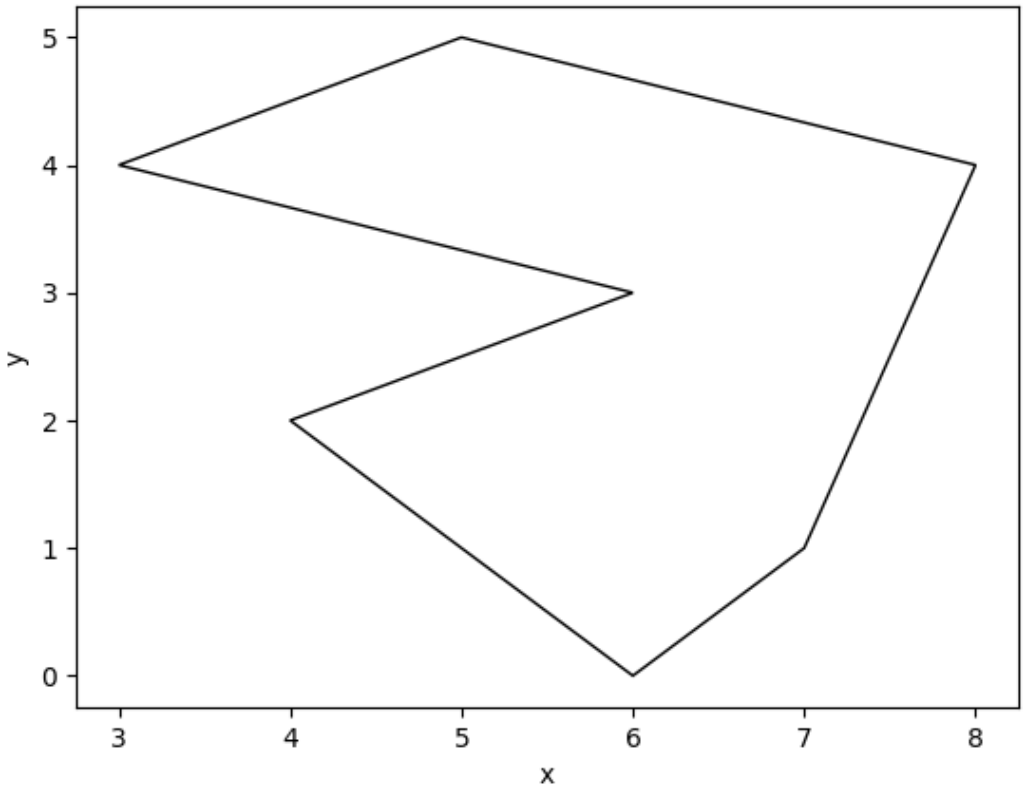
- sprawdzanie y-monotoniczności
- podział wierchołków na kategorie
- triangulacja wielokąta monotonicznego

Przykładowy wielokąt y-monotoniczny

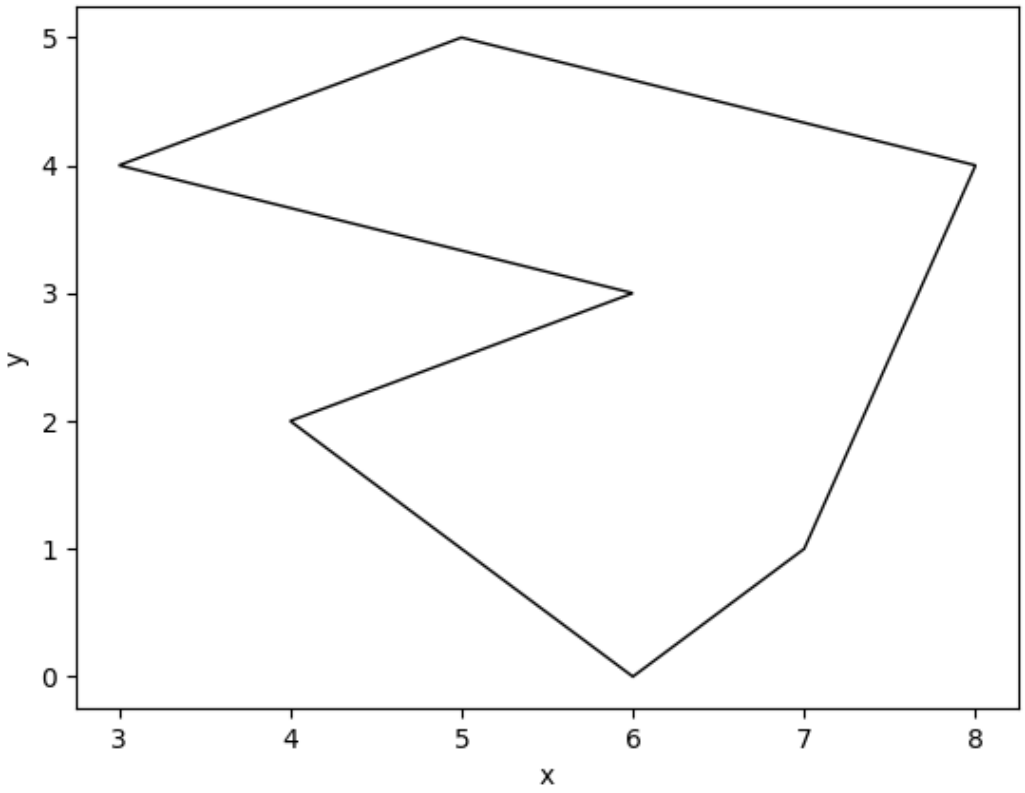
In [119…

```
draw_polygon(polygon_example_1)
```

Figure



Figure



Do tego celu wygeneruj wielokąt.

In [120…
```
polygon = [(5,5), (3,4), (6,3), (4,2), (6,0), (7,1), (10,4)]
```

Czy wielokąt jest y-monotoniczny?

Wielokąt jest monotoniczny, gdy jego wierzchołki mogą być ułożone w taki sposób, że jedna z jego współrzędnych (na przykład współrzędna x lub y, w zależności od układu współrzędnych) zawsze rośnie lub maleje wzdłuż kolejnych wierzchołków. Innymi słowy, dla każdej pary wierzchołków wielokąta (oprócz wierzchołka startowego i końcowego), jeden z punktów ma większą (lub mniejszą) wartość danej współrzędnej niż drugi punkt.

W praktyce, wielokąt monotoniczny może być łatwiej sortowany lub przetwarzany w pewnych algorytmach geometrycznych, ponieważ istnieje pewna kolejność, w jakiej wierzchołki pojawiają się wzdłuż danej osi (np. osi x lub y). Monotoniczność może ułatwić znajdowanie przecięć linii w takim wielokącie lub wykonywanie innych operacji geometrycznych. W tym zadaniu interesuje nas monotoniczność wielokąta wzdłuż osi y.

Ćw. Uzupełnij funkcję `is_y_monotonic`. Pamiętaj, aby sprawozdanie zawierało krótki opis działania tej funkcji.

In [121…
```
def is_y_monotonic(polygon):
    """
    Funkcja określa czy podana figura jest y-monotoniczna.
    :param polygon: tablica krotek punktów na płaszczyźnie euklidesowej p
    :return: wartość bool - true, jeśli wielokąt jest monotoniczny i fals
    """

    n = len(polygon)
    idx = 0
    for i in range(1,n):
        if polygon[idx][1] < polygon[i][1]:
            idx = i
    left = True
    for i in range(n+1):
        if left and polygon[(idx + i) % n][1] < polygon[(idx + i + 1) % n
            left = False
        elif not left and polygon[(idx + i) % n][1] > polygon[(idx + i +
            return i == n

    return False
```

Przeprowadź test poprawności powyższej funkcji.

In [122…
```
Test().runtest(1, is_y_monotonic)
```

```
Lab 3, task 1:
        Test 1: Passed
        Test 2: Passed
        Test 3: Passed
        Test 4: Passed
        Test 5: Passed
        Test 6: Passed
        Test 7: Passed
        Test 8: Passed
        Test 9: Passed
        Test 10: Passed
Result: 10/10
Time: 0.010s
```

Sprawdź monotoniczność swojego wielokątu.

In [123…
```
print(is_y_monotonic(polygon_example_1))
```

```
True
```

Podział wierzchołków na kategorie

Wierzchołki naszego wielokąta możemy podzielić na parę kategorii:

- początkowe, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma mniej niż 180 stopni. To wierzchołki, w których wielokąt zaczyna się monotoniczny spadek
- końcowe, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma mniej niż 180 stopni. To wierzchołki, w których monotoniczność wielokąta się zmienia, czyli na przykład zaczyna się monotoniczny wzrost, jeśli wcześniej był spadek, lub na odwrót.

Wierzchołki startowe i końcowe są ważne w kontekście algorytmów przetwarzania wielokątów monotonicznych, takich jak algorytmy dziel i zwyciężaj oraz triangulacji.

- dzielący, gdy obaj jego sąsiedzi leżą poniżej i kąt wewnętrzny ma więcej niż 180 stopni. To wierzchołki, które wyznaczają przekątne (linie łączące), tworzące trójkąty podczas triangulacji.
- łączący, gdy obaj jego sąsiedzi leżą powyżej i kąt wewnętrzny ma więcej niż 180 stopni. To wierzchołki, które są połączone liniami (przekątnymi) wewnątrz wielokąta, tworząc trójkąty.

Wierzchołki łączące i dzielące odgrywają kluczową rolę w procesie triangulacji wielokątów, pozwalając na podział figury na trójkąty w sposób bezkolizyjny.

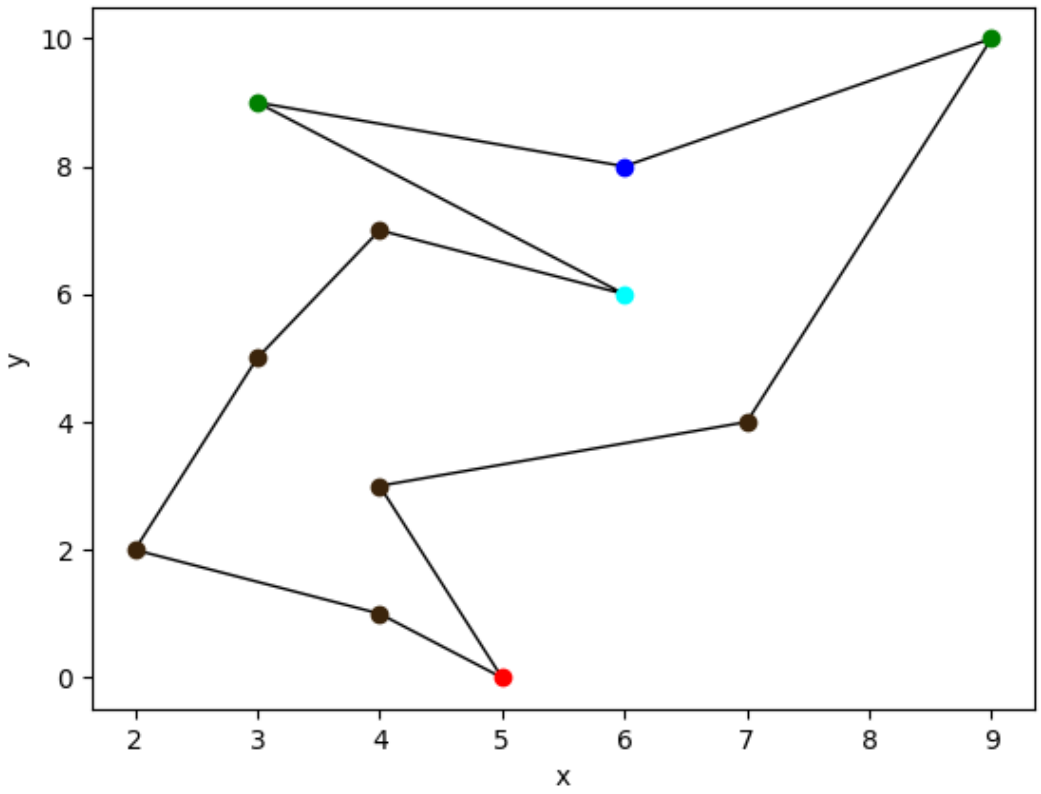
- prawdiłowy, pozostałe przypadki, jeden sąsiad powyżej drugi poniżej

Przykładowy wielokąt z pokolorowanymi wierzchołkami.

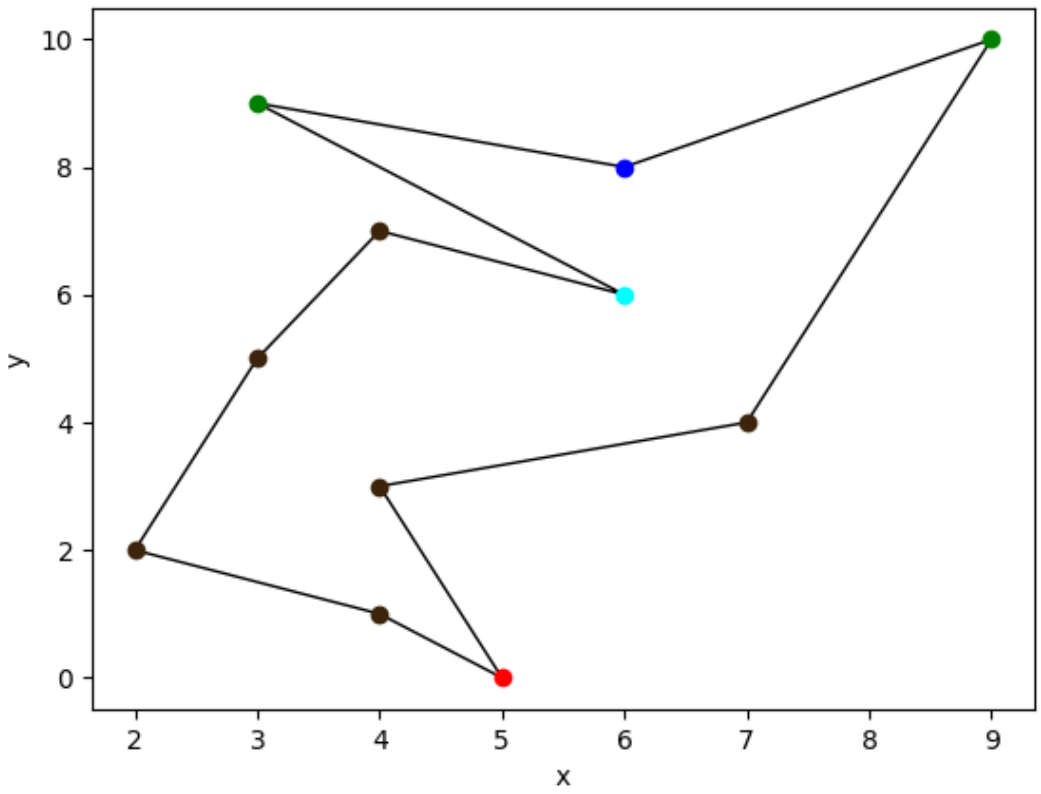
In [124…

```
draw_polygon_colors(polygon_example_2,polygon_example_colors)
```

Figure



Figure



Ćw. Uzupełnij funkcję `color_vertex` .

In [125…
```
def det(a, b, c):
    return (a[0] - c[0]) * (b[1] - c[1]) - (b[0] - c[0]) * (a[1] - c[1])
```

In [126…
```
def color_vertex(polygon):
    """

    Funkcja dzieli wierzchołki na kategorie i przypisuje wierzchołkom odp
    :param polygon: tablica krotek punktów na płaszczyźnie euklidesowej p
    :return: tablica o długości n, gdzie n = len(polygon), zawierająca cy
    """
    y = 1
    n = len(polygon)
    T= [0] * n
    for i in range(n):
        prev = polygon[i-1]
        next = polygon[(i+1)%n]
        if polygon[i][y] > next[y] and polygon[i][y] > prev[y] and det(pr
            T[i] = 0
        elif polygon[i][y] < next[y] and polygon[i][y] < prev[y] and det(
            T[i] = 1
        elif polygon[i][y] < next[y] and polygon[i][y]< prev[y] and det(p
            T[i] = 2
        elif polygon[i][y] > next[y] and polygon[i][y] > prev[y] and det(
            T[i] = 3
        else:
            T[i] = 4
    return T
```

Przeprowadź test poprawności powyższej funkcji.

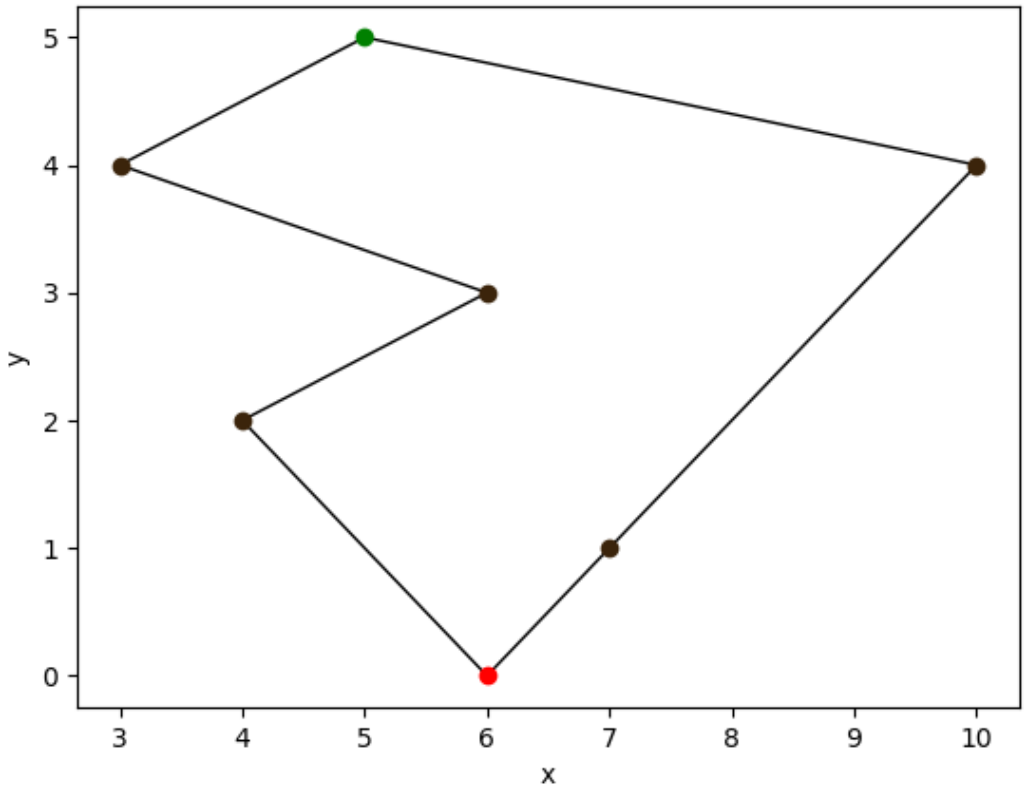
In [127…
```
Test().runtest(2, color_vertex)
```

```
Lab 3, task 2:
        Test 1: Passed
        Test 2: Passed
        Test 3: Passed
        Test 4: Passed
        Test 5: Passed
        Test 6: Passed
        Test 7: Passed
        Test 8: Passed
        Test 9: Passed
        Test 10: Passed
Result: 10/10
Time: 0.006s
```

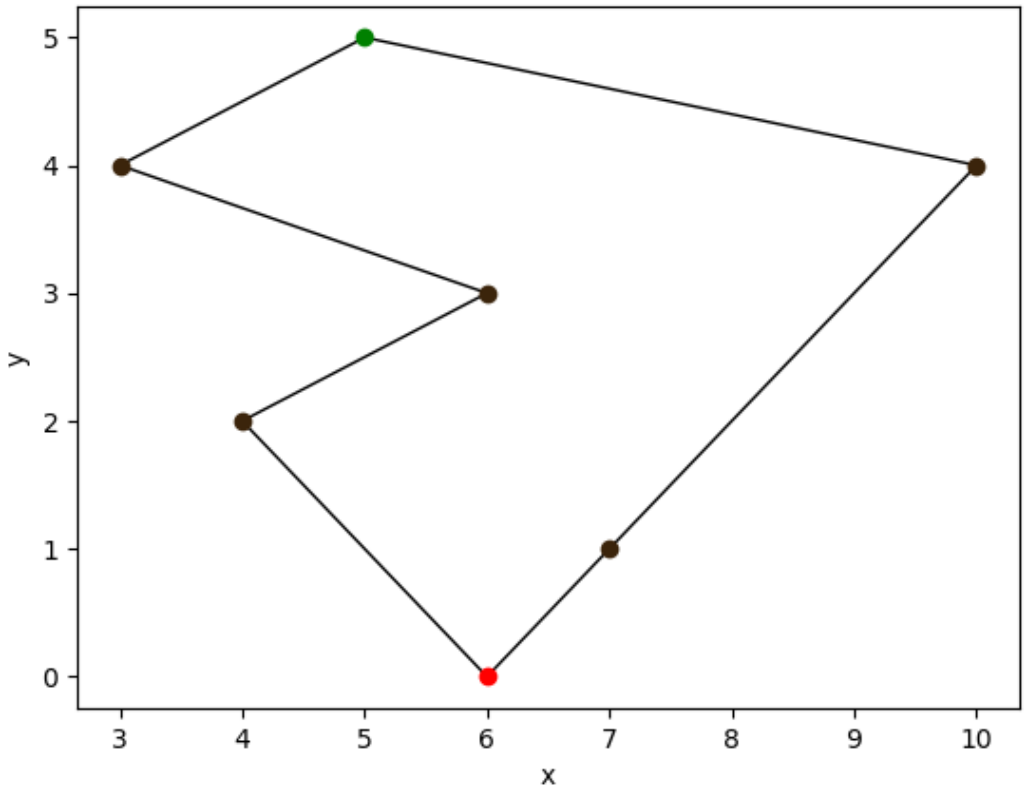
Zwizualizuj swój wielokąt z pokolorowanymi według kategorii wierzchołkami.

In [128…
```
colors = color_vertex(polygon)
draw_polygon_colors(polygon,colors)
```

Figure



Figure



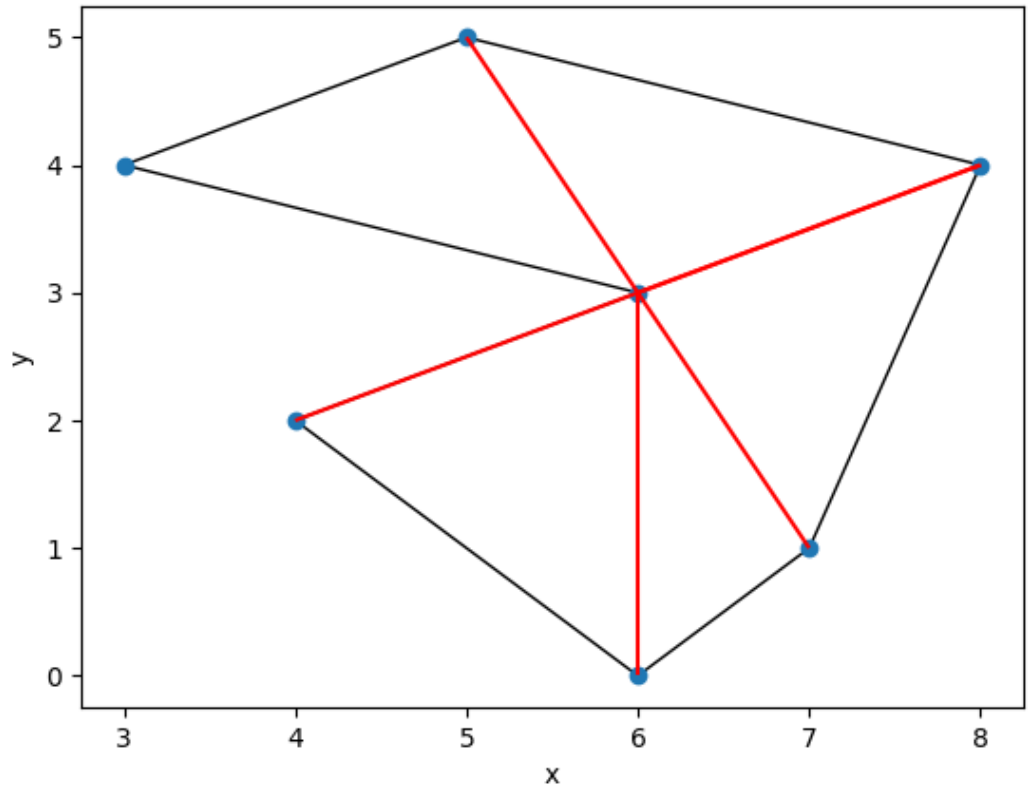
Triangulacja wielokąta monotonicznego

Triangulacja wielokąta monotonicznego to proces podziału wielokąta monotonicznego na trójkąty poprzez dodawanie przekątnych (linii łączących wierzchołki), które nie przecinają się wewnętrznie.

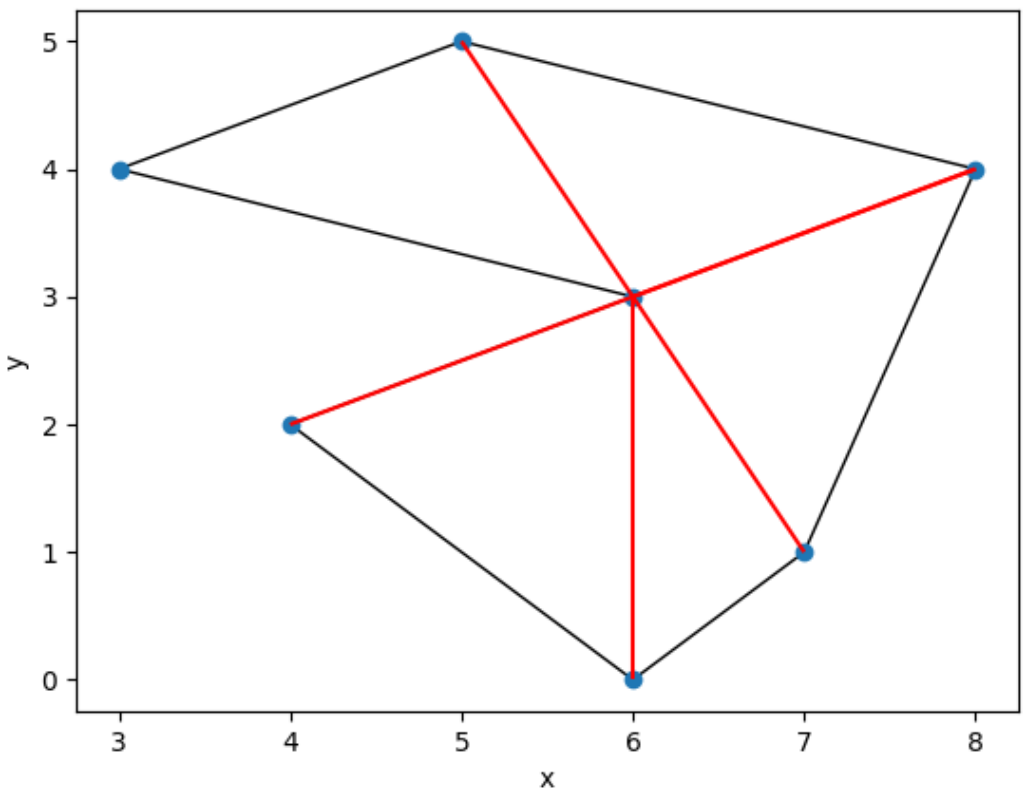
Przykładowy wielokąt podzielony na trójkąty.

In [129…
```
draw_polygon_tri(polygon_example_1,polygon_example_tri)
```

Figure



Figure



Ćw. Uzupełnij funkcję `triangulation`. Wykorzystaj algorytm opisany na wykładzie.

In [130…

```
def chains(polygon):
    """ the lowest point is not in left or right chain """
    n = len(polygon)
    idx = 0
    for i in range(1,n):
        if polygon[idx][1] < polygon[i][1]:
            idx = i
    L = set()
```

```
    R = set()
    left = True
    for i in range(n):
        if left and polygon[(idx + i) % n][1] < polygon[(idx + i + 1) % n
            left = False
        elif left:
            L.add(polygon[(idx + i) % n])
        else:
            R.add(polygon[(idx + i) % n])
    return L, R

def different_chains(p, S, L, R):
    return (p not in L and S[-1] in L) or (p not in R and S[-1] in R)

def save_indices(polygon):
    return {p: i for i, p in enumerate(polygon)}

def det(a, b, c):
    return (a[0] - c[0]) * (b[1] - c[1]) - (b[0] - c[0]) * (a[1] - c[1])

def is_triangle_inside(p1, p2, p3, L, R):
    if p3 in L:
        return det(p1, p2, p3) > 0
    else:
        return det(p1, p2, p3) < 0

def triangulation(polygon):
    """
    Funkcja dokonuje triangulacji wielokąta monotonicznego.
    :param polygon: tablica krotek punktów na płaszczyźnie euklidesowej p
    :return: tablica krotek dodawanych po kolei przekątnych np: [(1,5),(2
    """
    n = len(polygon)
    L, R = chains(polygon)
    idx = save_indices(polygon)
    polygon = sorted(polygon, key=lambda p: p[1], reverse=True)

    P = []
    S = [polygon[0], polygon[1]]
    for i in range(2, n):
        if different_chains(polygon[i], S, L, R):
            for p in S[1:]:
                P.append([idx[p], idx[polygon[i]]])
            S = [S[-1], polygon[i]]
        else:
            while len(S) >= 2:
                if is_triangle_inside(S[-2], S[-1], polygon[i], L, R):
                    P.append([idx[S[-2]], idx[polygon[i]]])
                    S.pop()
                else:
                    break
            S.append(polygon[i])
    return P[:-1]
```

Jakich struktur można użyć do przechowywania wielokątu, oraz utworzonej triangulacji? Uzasadnij wybór struktury w swoim algorytmie

ODPOWIEDŹ:

Przeprowadź testy poprawności powyższej funkcji.

In [131…
```
Test().runtest(3, triangulation)
```

```
Lab 3, task 3:
        Test 1: Passed
        Test 2: Passed
        Test 3: Passed
        Test 4: Passed
        Test 5: Passed
        Test 6: Passed
        Test 7: Passed
        Test 8: Passed
        Test 9: Passed
        Test 10: Passed
Result: 10/10
Time: 0.011s
```

Zwizualizuj powstały wielokąt podzielony na trójkąty.

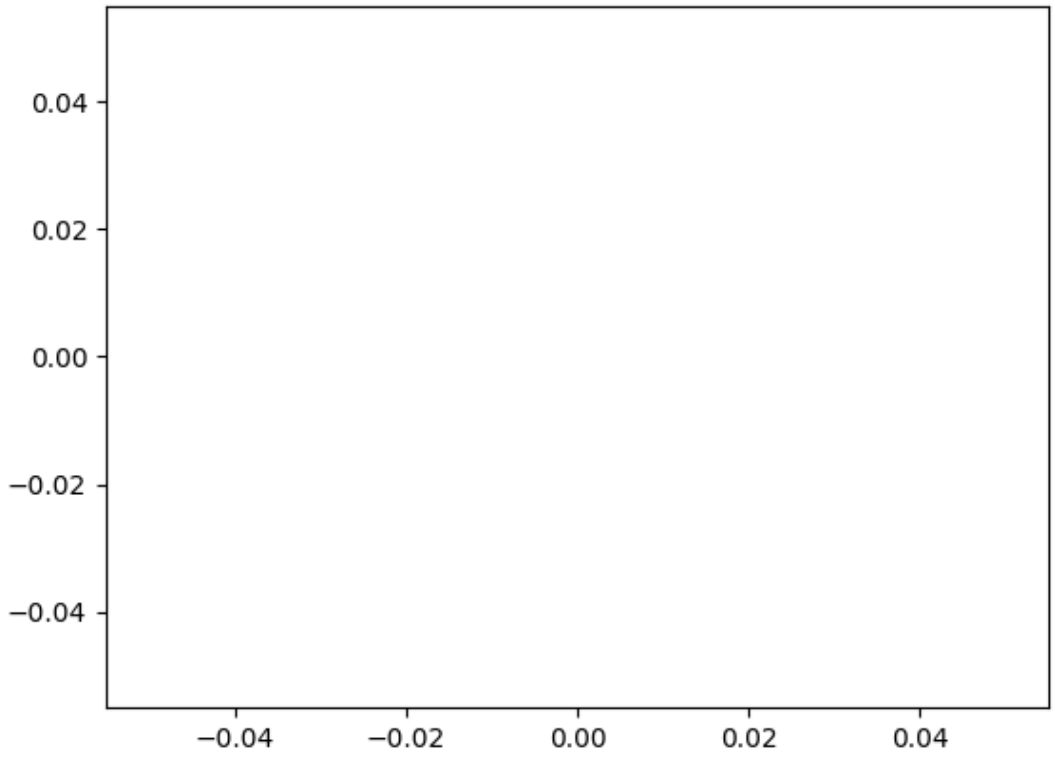
In [132…
```
import json
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.collections as mcoll
import matplotlib.widgets as mwdg

from matplotlib.backend_bases import MouseEvent
from matplotlib.patches import Polygon

BLUE = u'#1f77b4'


class Callback:
    def __init__(self, ax):
        self.added_points = []
        self.ax = ax

    def on_click(self, event):
        if event.inaxes != self.ax:
            return
        new_point = (event.xdata, event.ydata)
        self.added_points.append(new_point)
        print(new_point)
        self.draw(autoscaling=False)

    def draw_points(self):
        if self.added_points:
            self.ax.scatter(*zip(*(np.array(self.added_points))))

    def draw_lines(self):
        if len(self.added_points) >= 2:
            # lines = []
            for i in range(len(self.added_points)):
                self.ax.plot([self.added_points[i-1][0], self.added_point
```

```
                                [self.added_points[i-1][1], self.added_point

    def draw_polygon(self):
        if len(self.added_points) >= 3:
            p = Polygon(self.added_points, alpha=0.3)
            self.ax.add_patch(p)

    def draw(self, autoscaling=True):
        if not autoscaling:
            xlim = self.ax.get_xlim()
            ylim = self.ax.get_ylim()
        self.ax.clear()
        self.draw_points()
        self.draw_lines()
        self.draw_polygon()
        self.ax.autoscale(autoscaling)
        if not autoscaling:
            self.ax.set_xlim(xlim)
            self.ax.set_ylim(ylim)
        plt.draw()


class InputVisualizer(Visualizer):
    def __init__(self):
        super().__init__()
        self.callback = None
        plt.close()
        self.fig = plt.figure()
        self.ax = plt.axes(autoscale_on=False)

    def get_added_points(self):
        plt.close()
        return self.callback.added_points

    def input(self):
        self.callback = Callback(self.ax)
        self.fig.canvas.mpl_connect('button_press_event', self.callback.o
        self.callback.draw()
        plt.show()
```

In [133…

```
%matplotlib widget

vis = InputVisualizer()
vis.input()
```

Figure



In [134…
```
polygon = vis.get_added_points()
```

Polygon_1

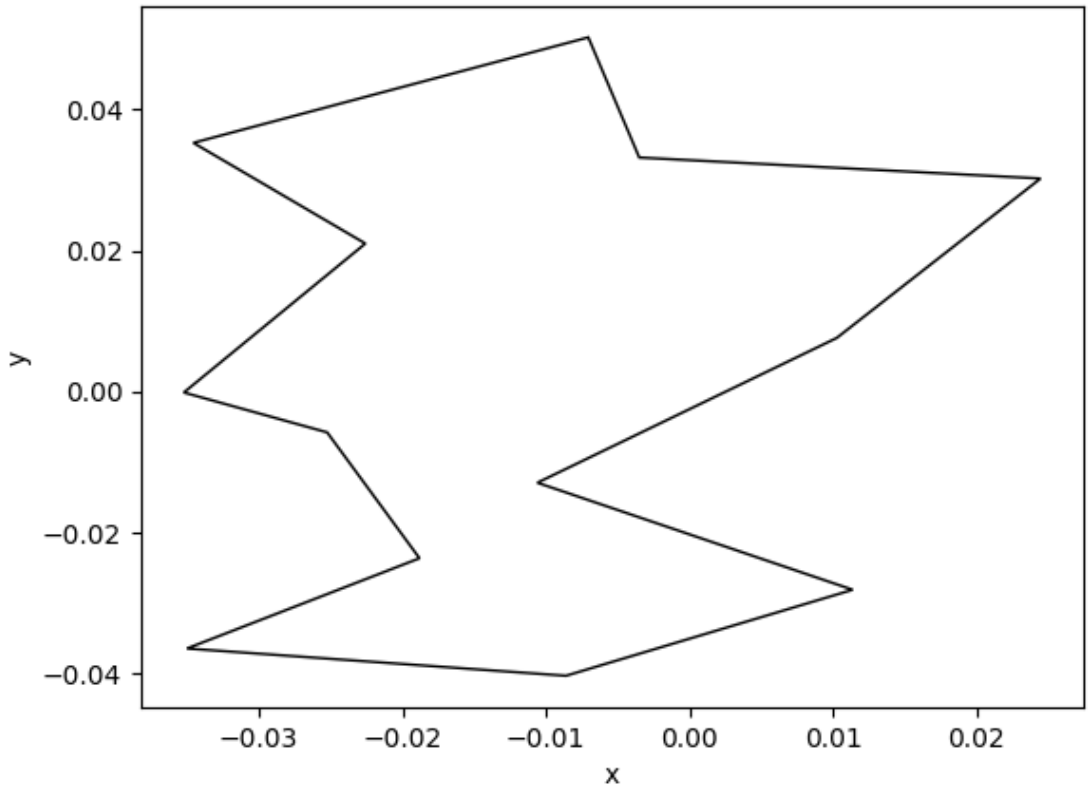
In [135…
```
polygon_1 = [(-0.007046995470600739, 0.050245869582865726), (-0.034546995
```

In [136…
```
print(is_y_monotonic(polygon_1))
```

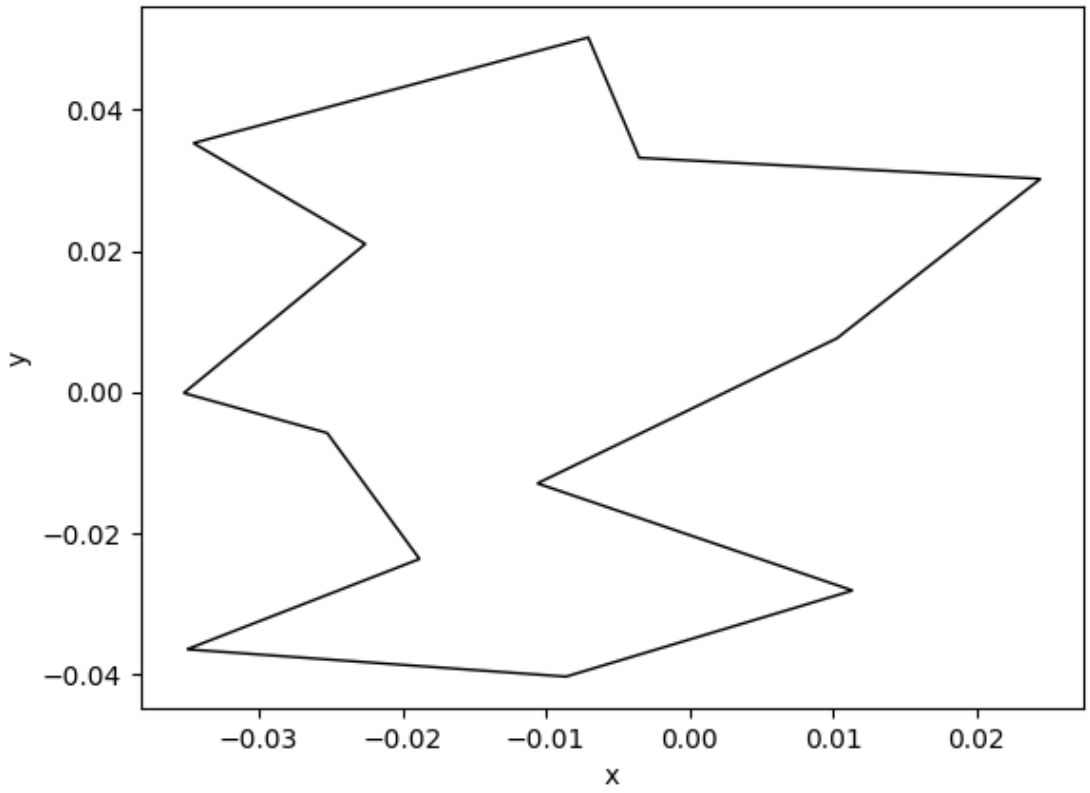
```
True
```

In [137…
```
draw_polygon(polygon_1)
```

Figure



Figure



polygon_2

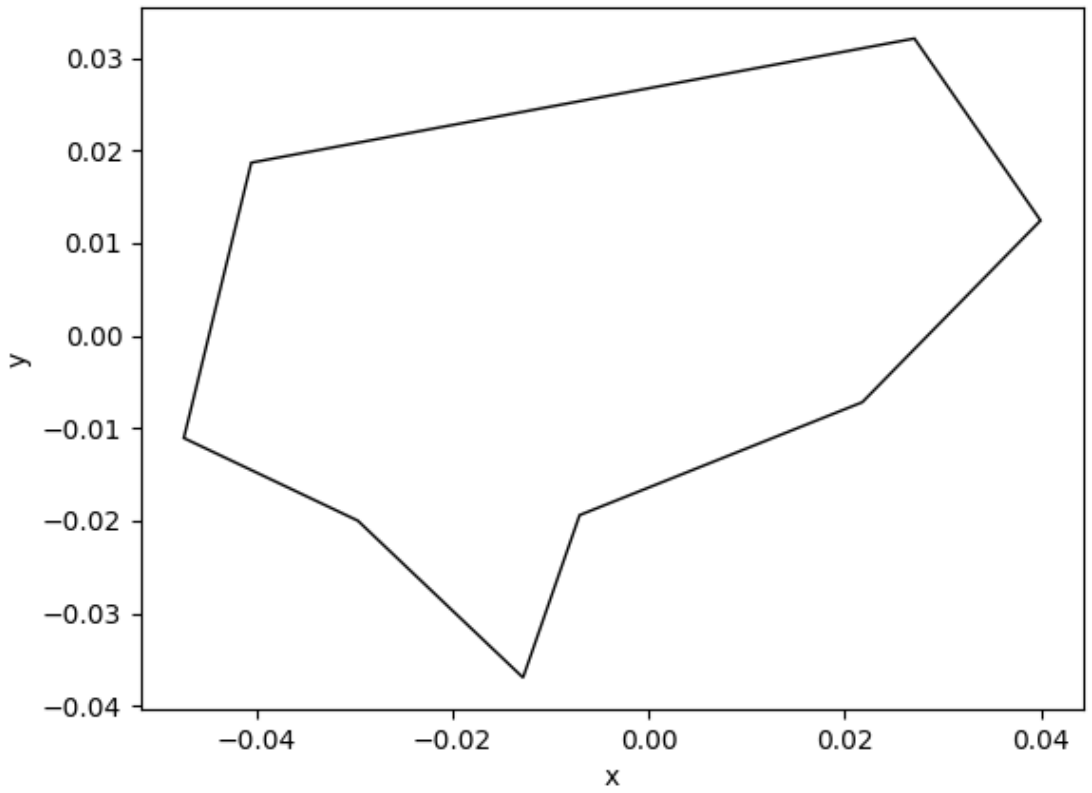
In [138…
```
polygon_2 = [(0.02710623033585087, 0.03211816478085215), (-0.040534898696
```

In [139…
```
print(is_y_monotonic(polygon_2))
```

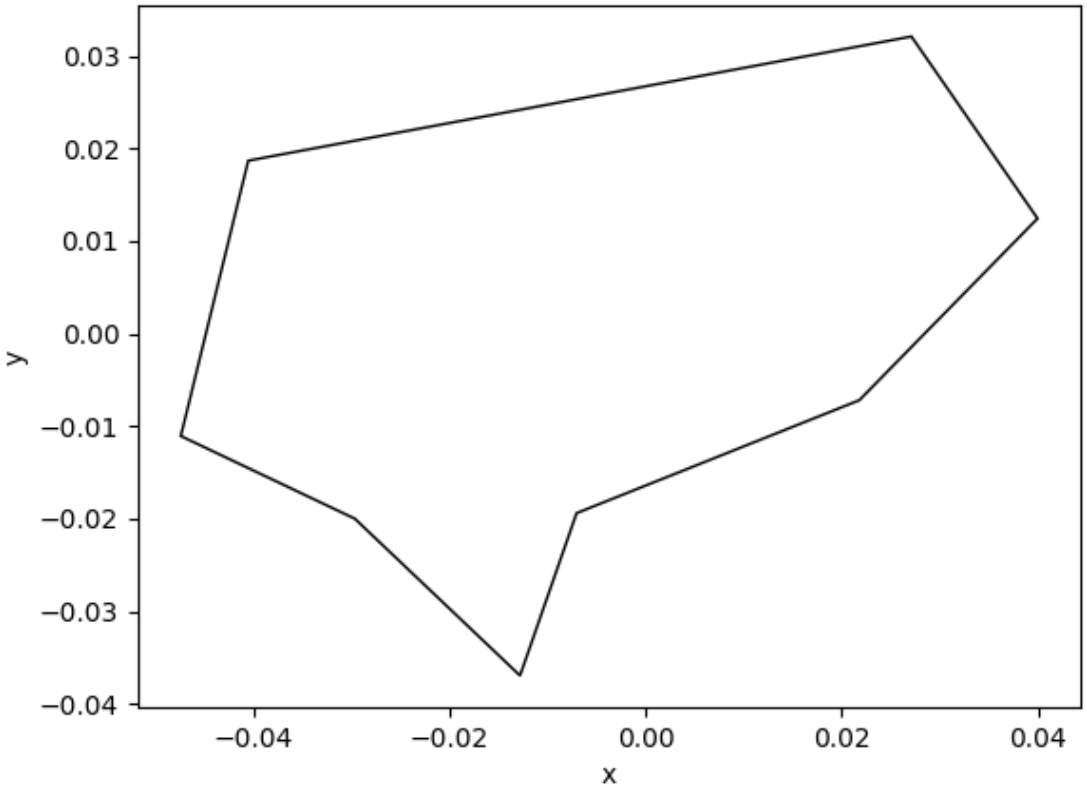
```
True
```

In [140…
```
draw_polygon(polygon_2)
```

Figure



Figure



polygon_3

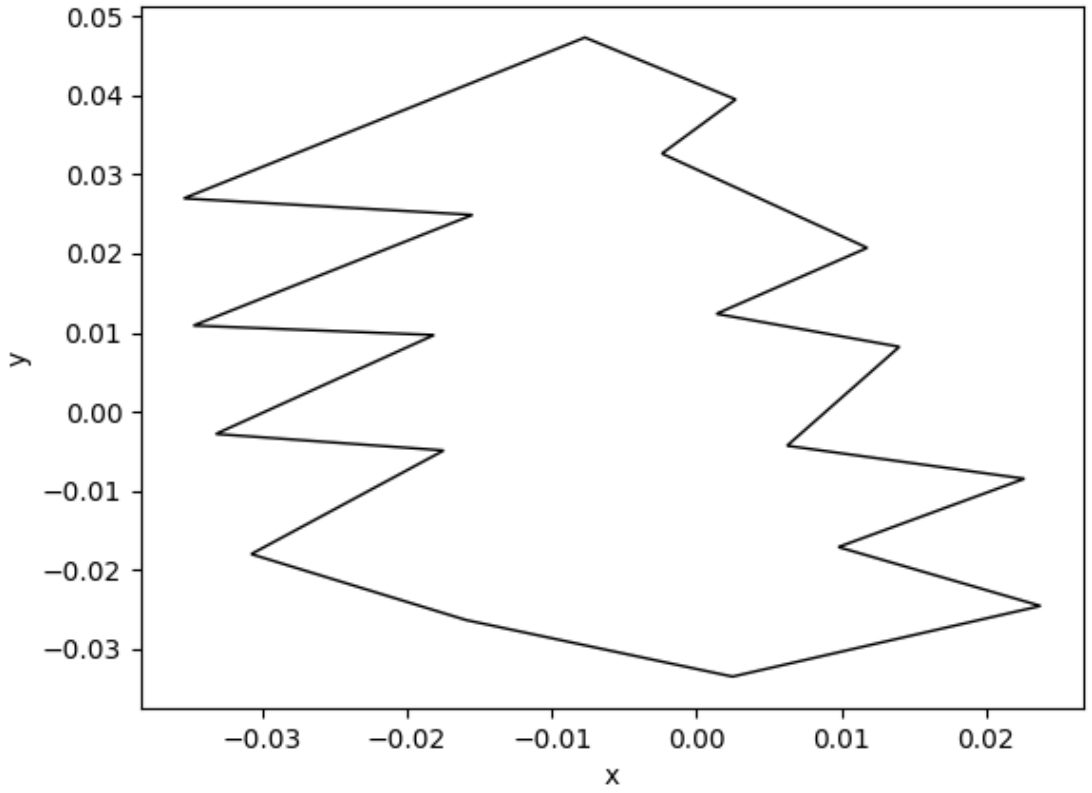
In [141…
```
polygon_3 = [(-0.007712318051245906, 0.04726967910667525), (-0.0354340922
```

In [142…
```
print(is_y_monotonic(polygon_3))
```

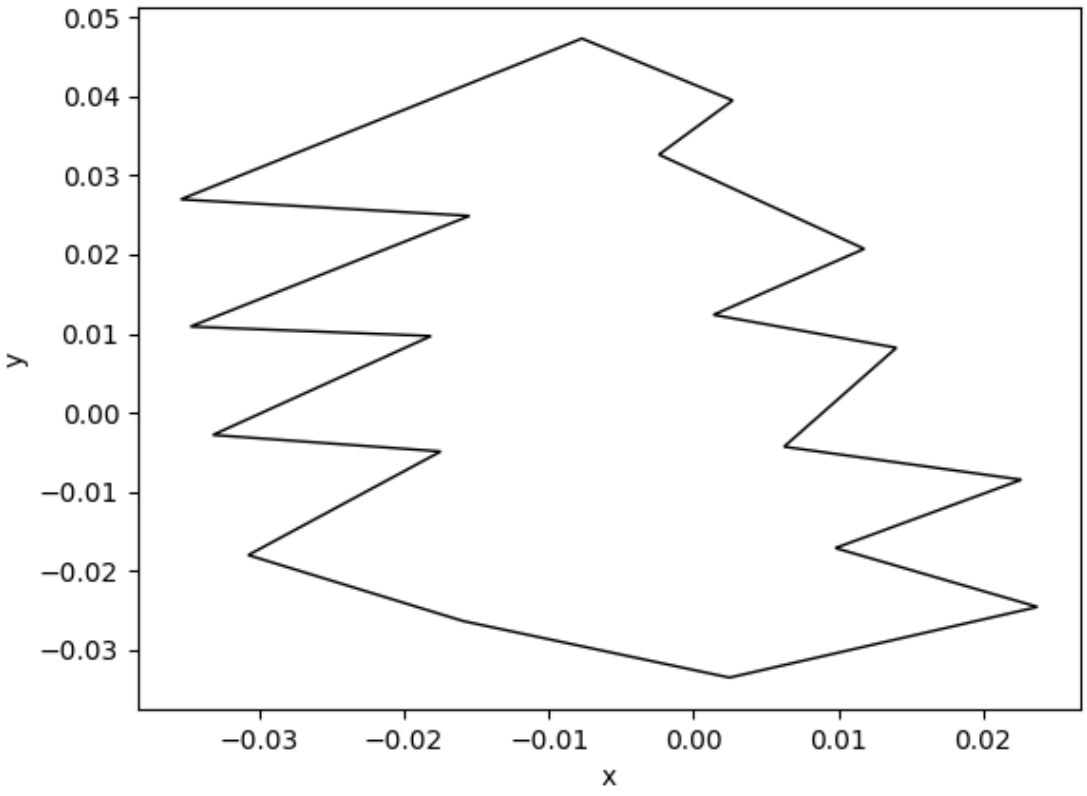
```
True
```

```
In [143…  draw_polygon(polygon_3)
```

Figure



Figure



polygon4

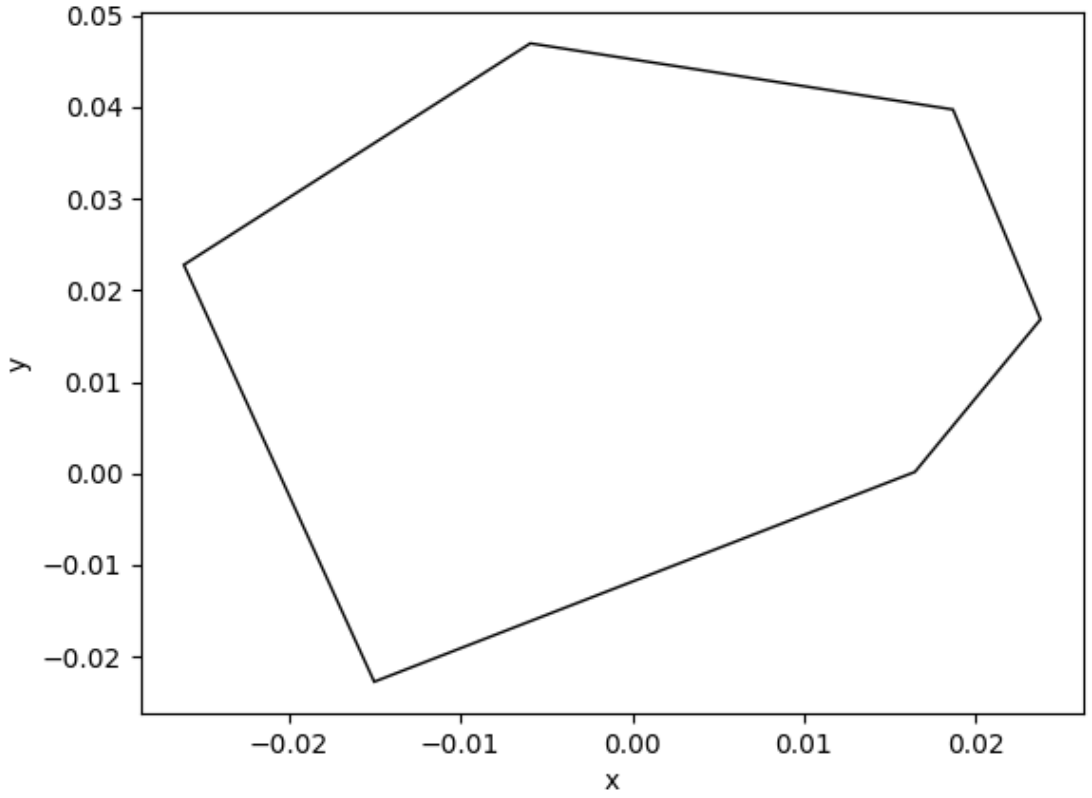
In [144…
```
polygon_4 = [(-0.005938124502858813, 0.0469720600590562), (-0.02611957611
```

In [145…
```
print(is_y_monotonic(polygon_4))
```

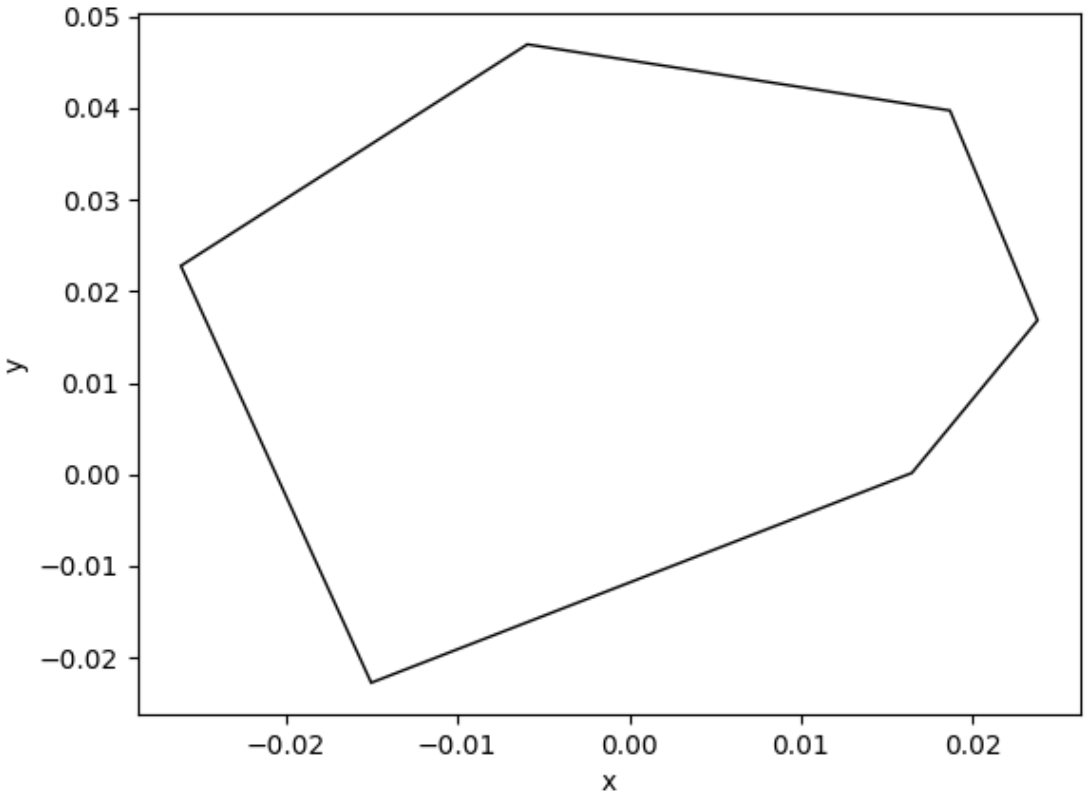
```
True
```

In [146…
```
draw_polygon(polygon_4)
```

Figure



Figure

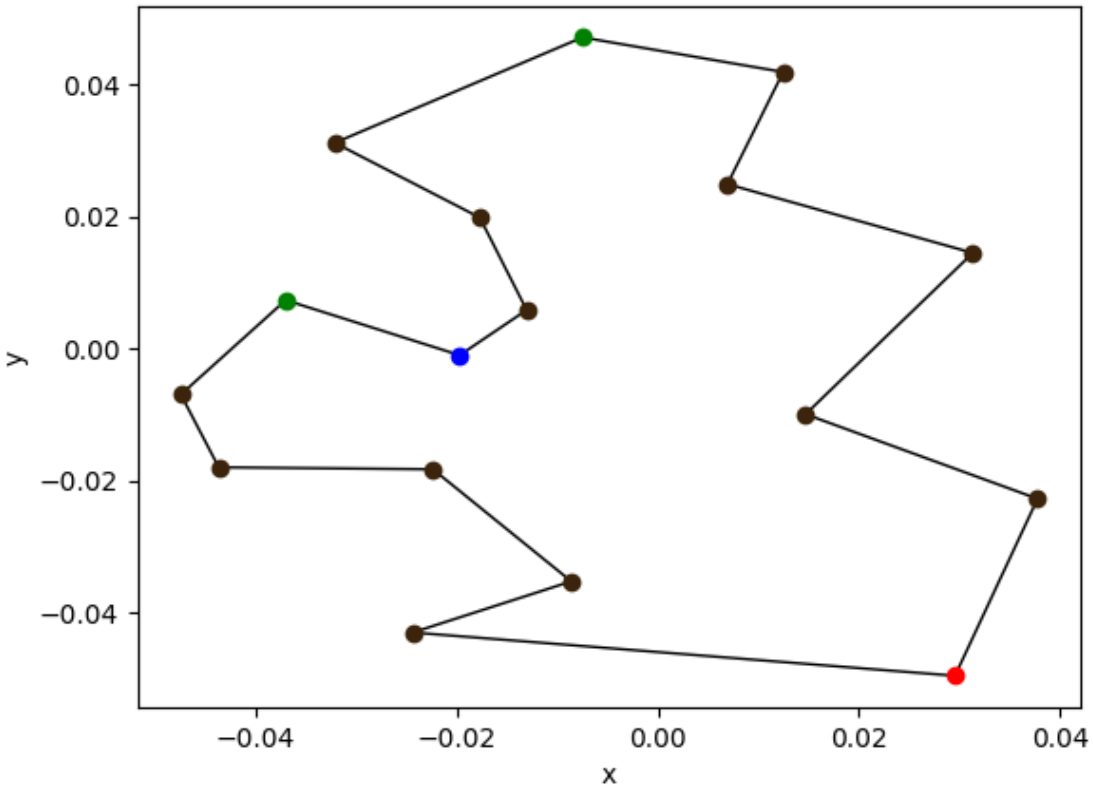


Podział wierzchołków

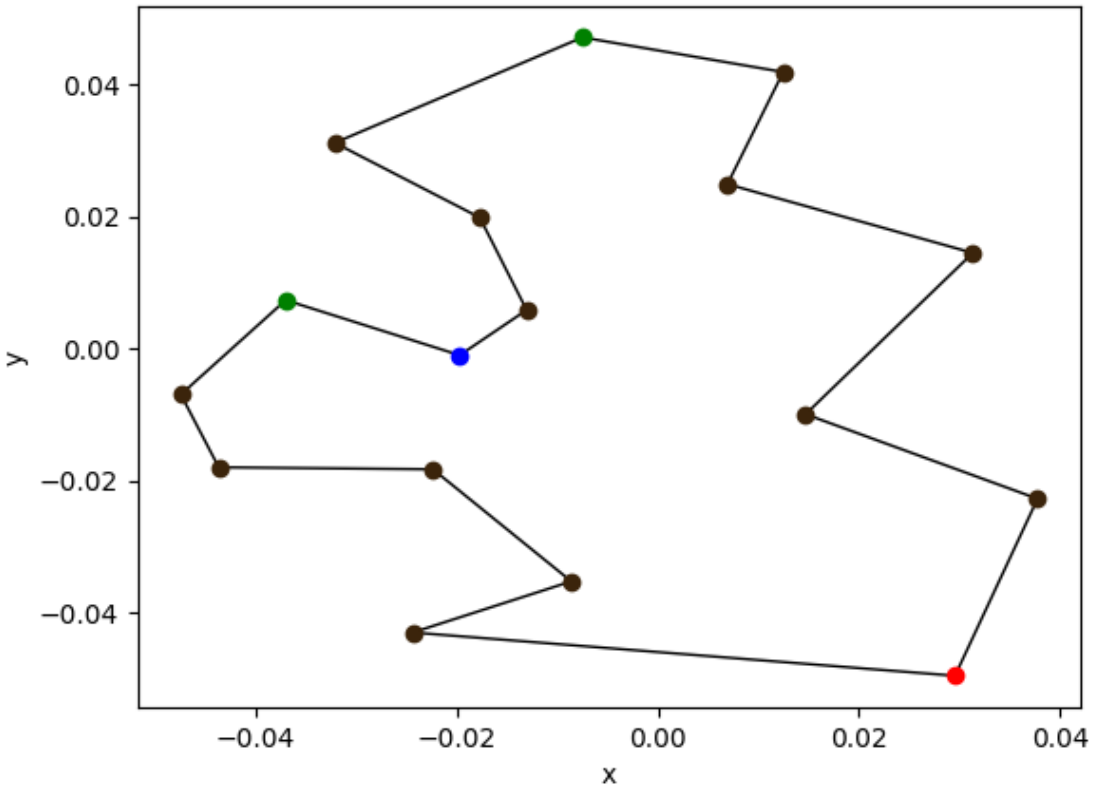
In [147…
```
p1 = [(-0.0074905438576975125, 0.04708028619939637), (-0.0321074793415684
```

In [148…
```
colors = color_vertex(p1)
draw_polygon_colors(p1,colors)
```

Figure



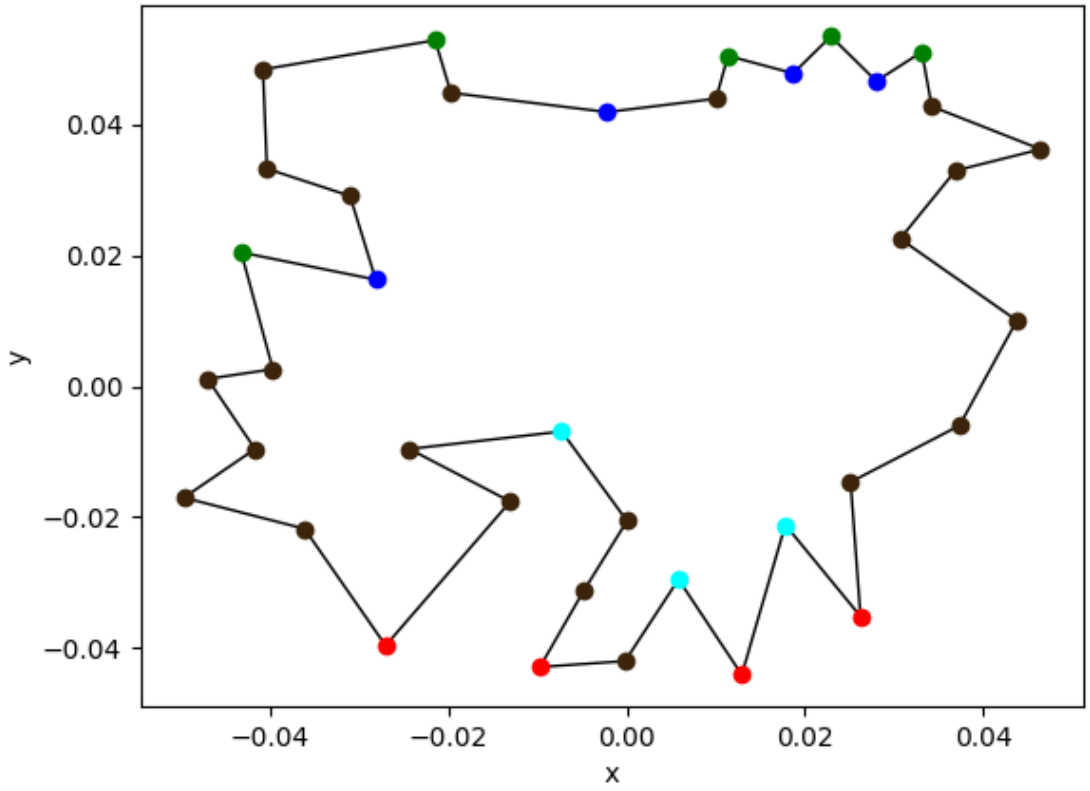
Figure



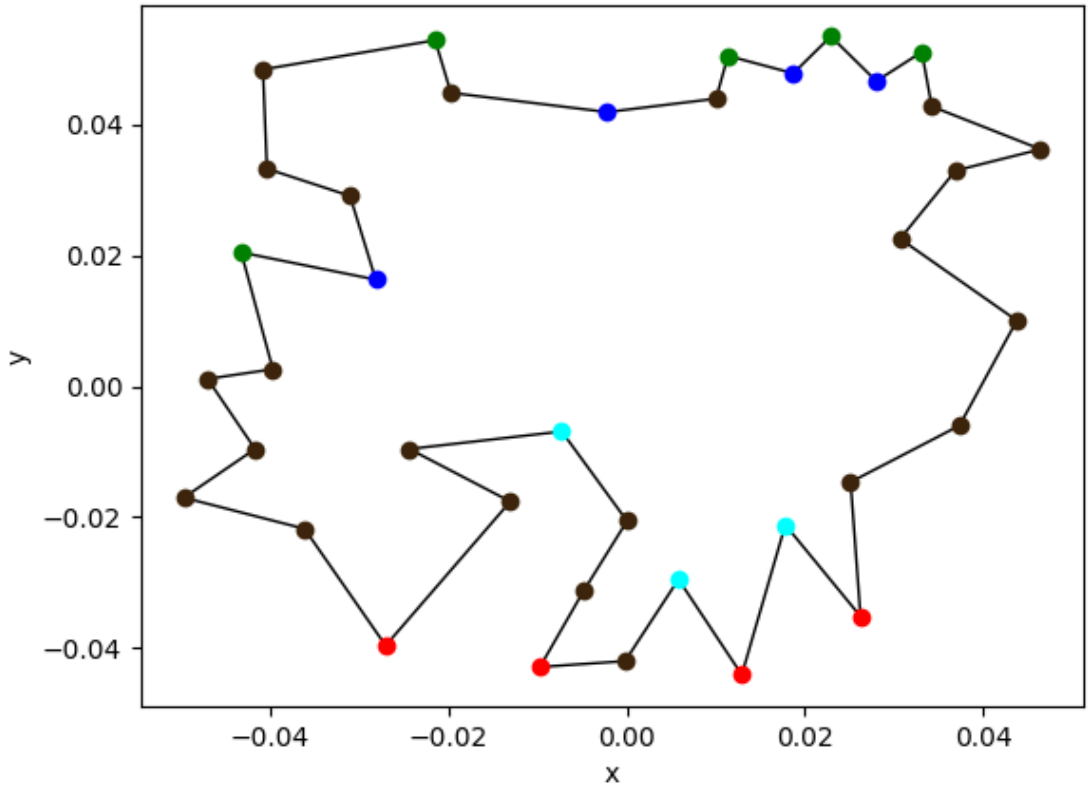
In [149…
```
p2 = [(-0.0021679632125362197, 0.041939593352280716), (-0.019688124502858
```

In [150…
```
colors = color_vertex(p2)
draw_polygon_colors(p2,colors)
```

Figure



Figure

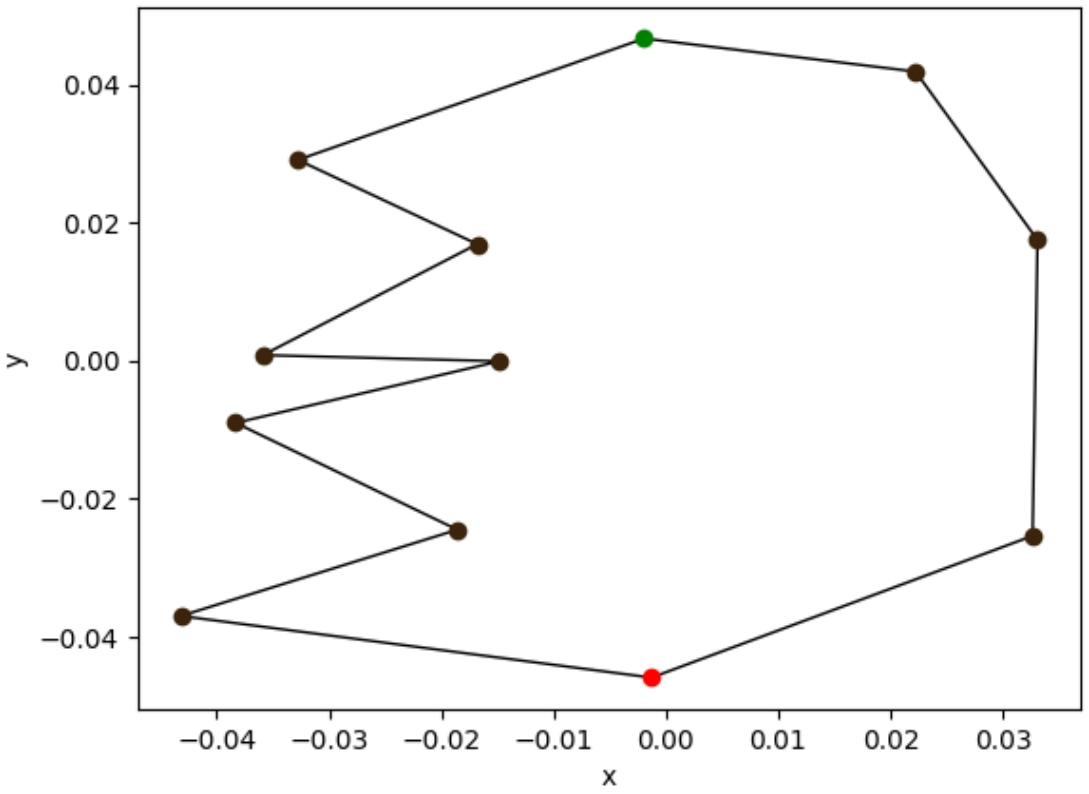


polygon3

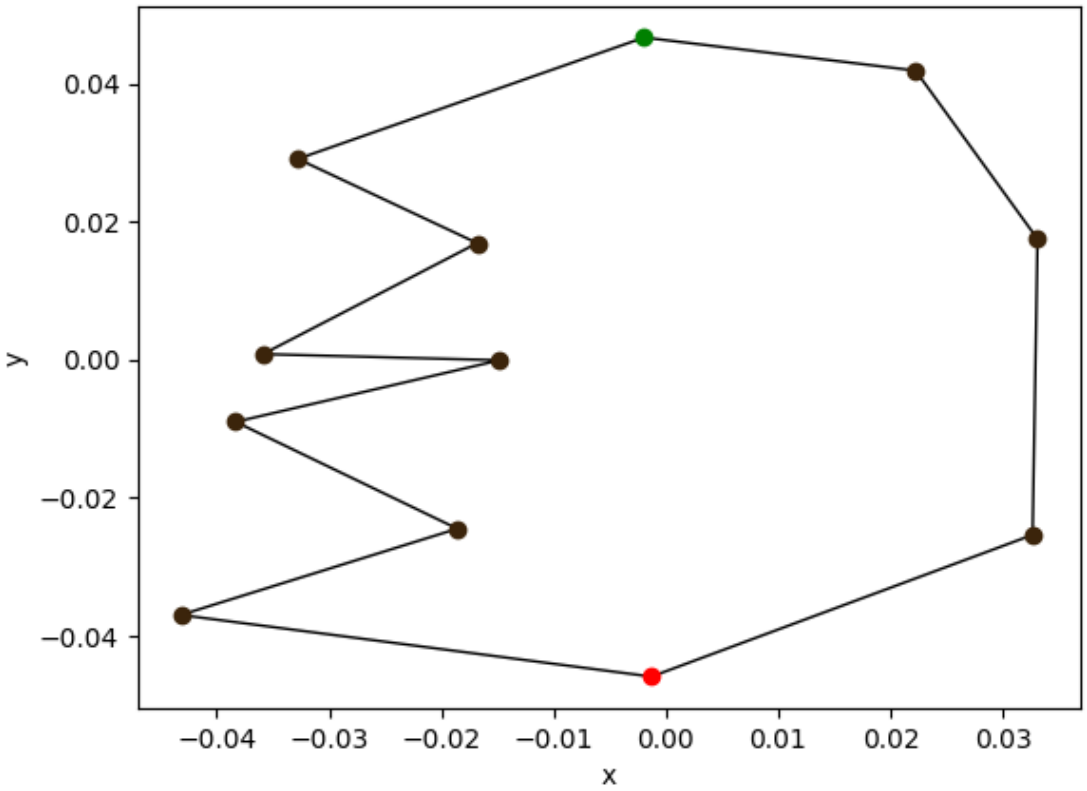
In [151…
```
p3 = [(-0.00194618901898784, 0.04670149811418549), (-0.03277280192221365,
```

In [152…
```
colors = color_vertex(p3)
draw_polygon_colors(p3,colors)
```

Figure



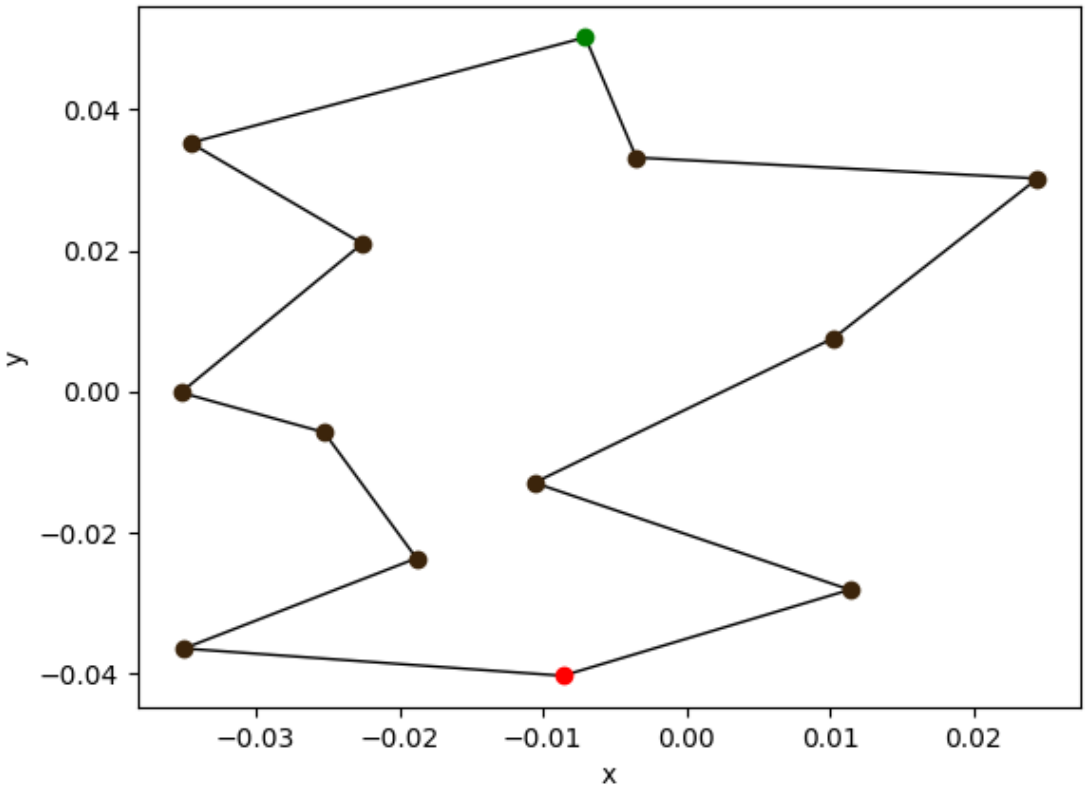
Figure



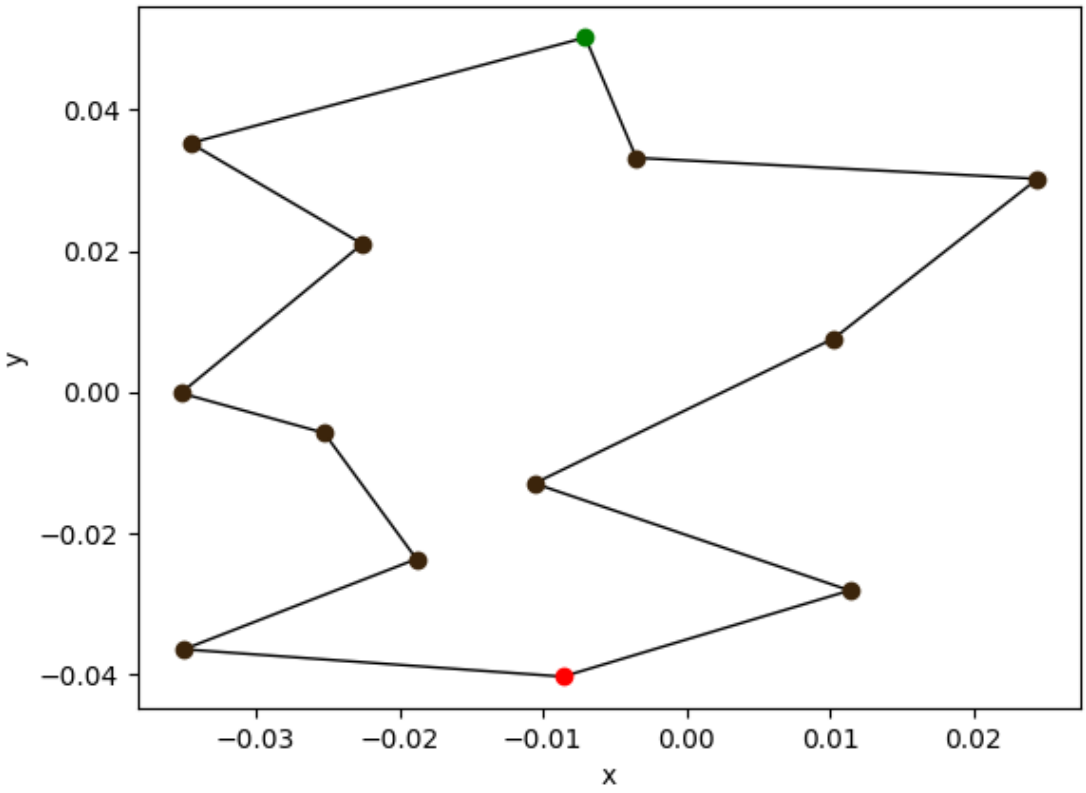
In [164…

```
colors = color_vertex(polygon_1)
draw_polygon_colors(polygon_1,colors)
```

Figure



Figure



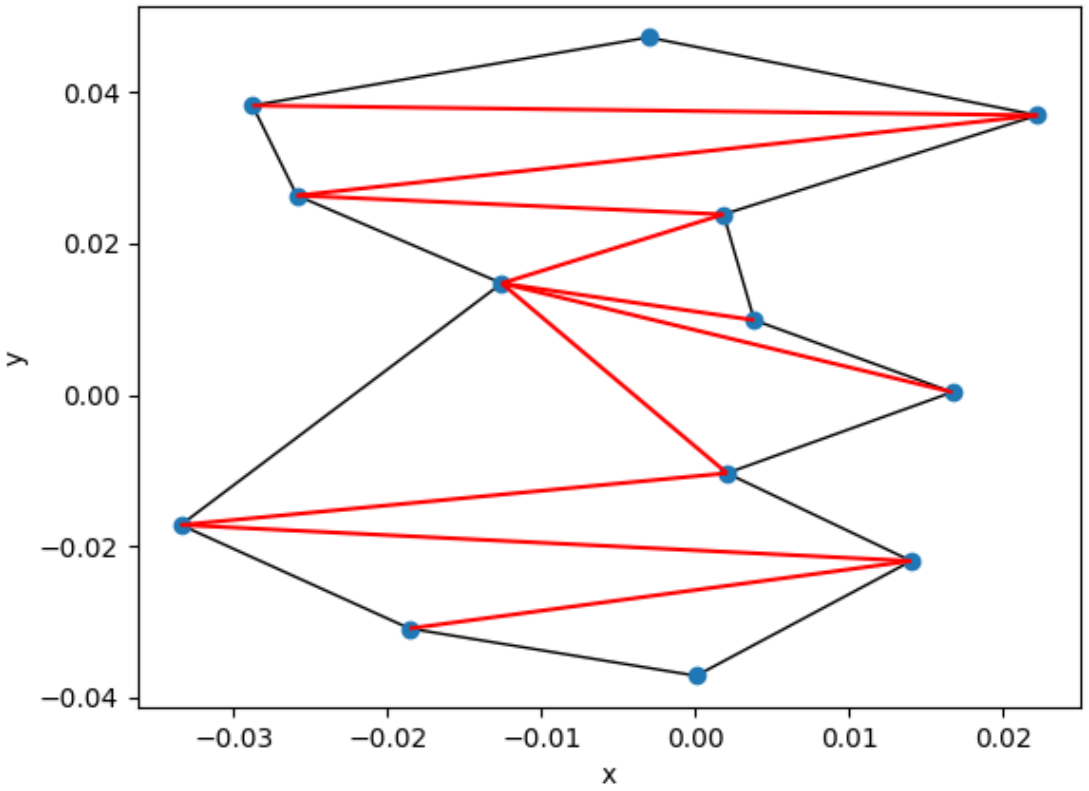
triangulacja

polygon1

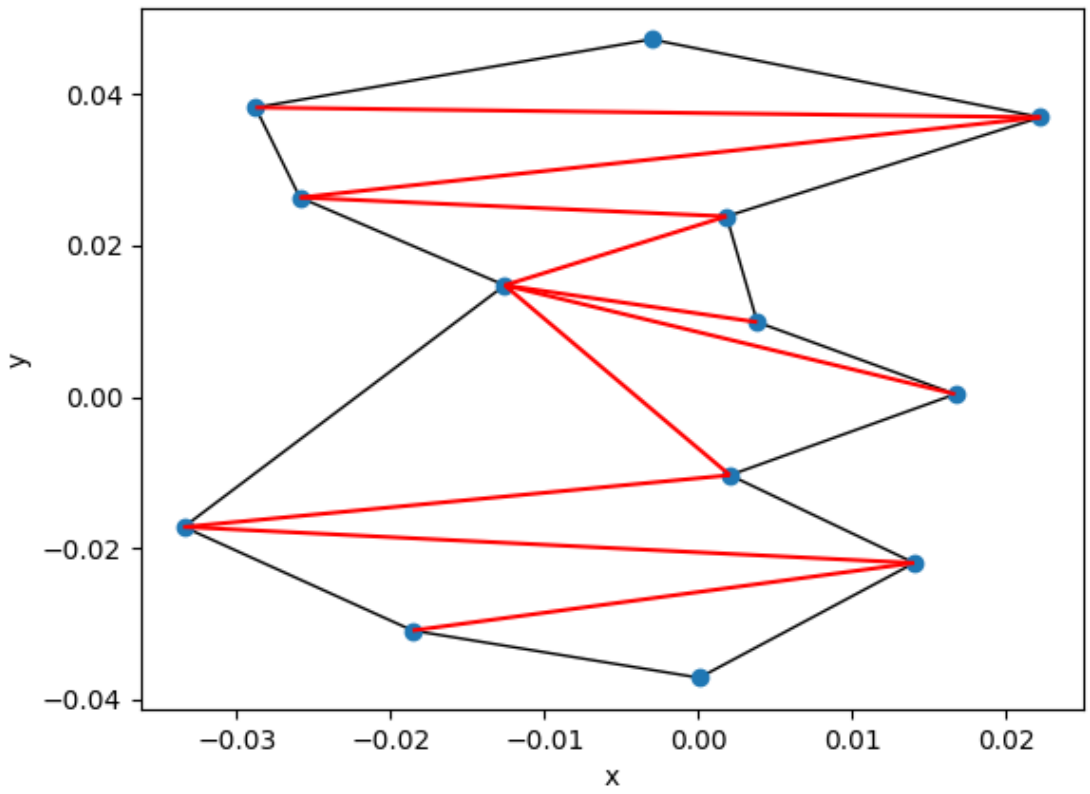
In [165…
```
t1 = [(-0.00305505998672978, 0.04726967910667525), (-0.028780866438342684
```

In [166…
```
assert is_y_monotonic(t1), 'Polygon is not y-monotonic'
tri = triangulation(t1)
draw_polygon_tri(t1,[[t1[diag[0]], t1[diag[1]]] for diag in tri])
```

Figure



Figure



In [ ]:

polygon2

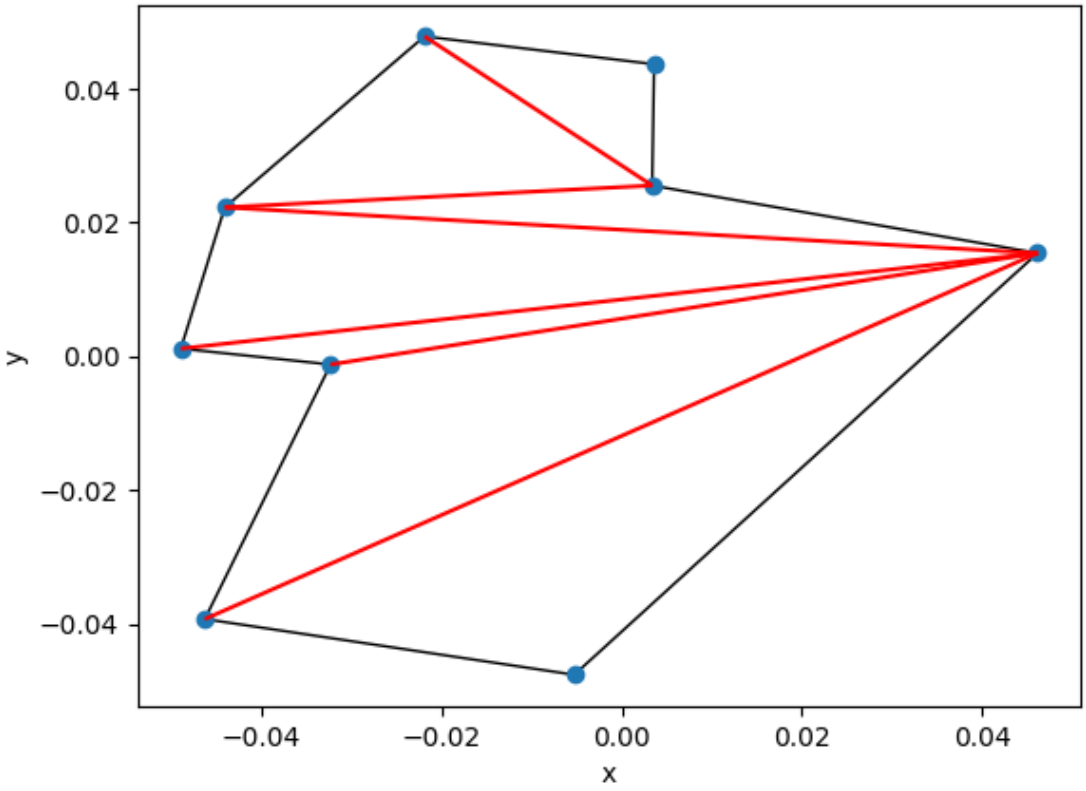
In [167…

```
t2 = [(-0.021905866438342685, 0.047891974304661675), (-0.0440832857931813
```

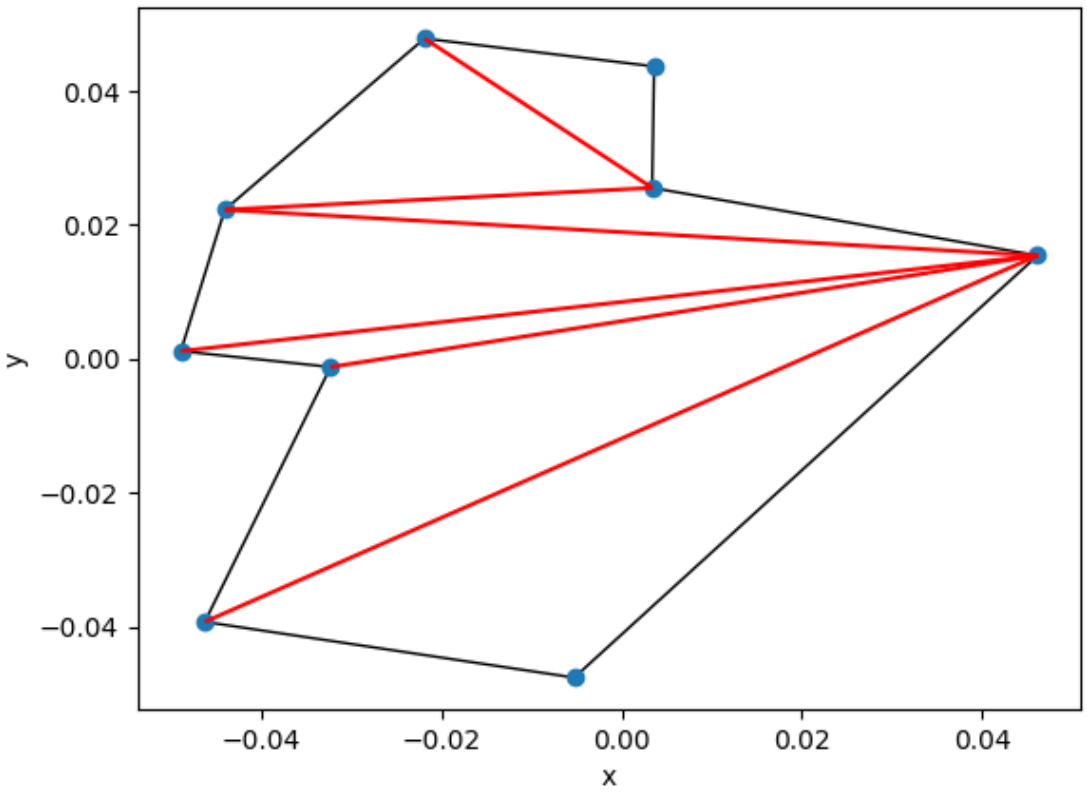
In [168…

```
assert is_y_monotonic(t2), 'Polygon is not y-monotonic'
tri = triangulation(t2)
draw_polygon_tri(t2,[[t2[diag[0]], t2[diag[1]]] for diag in tri])
```

Figure



Figure

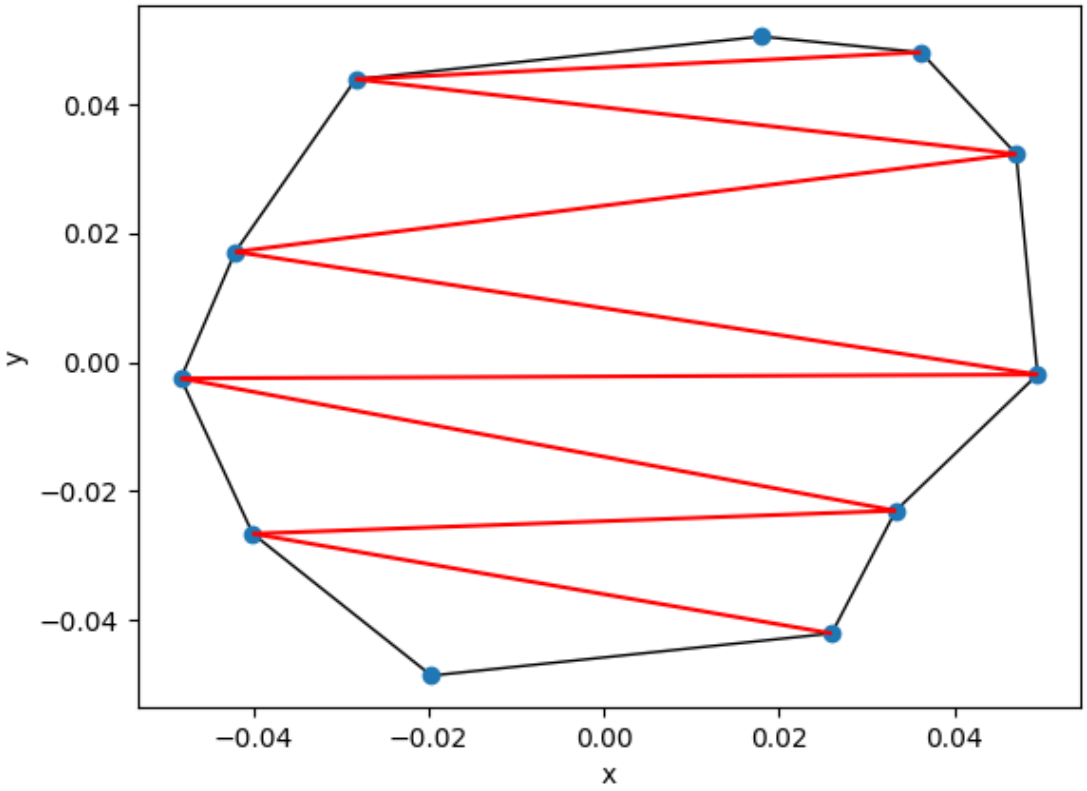


polygon3

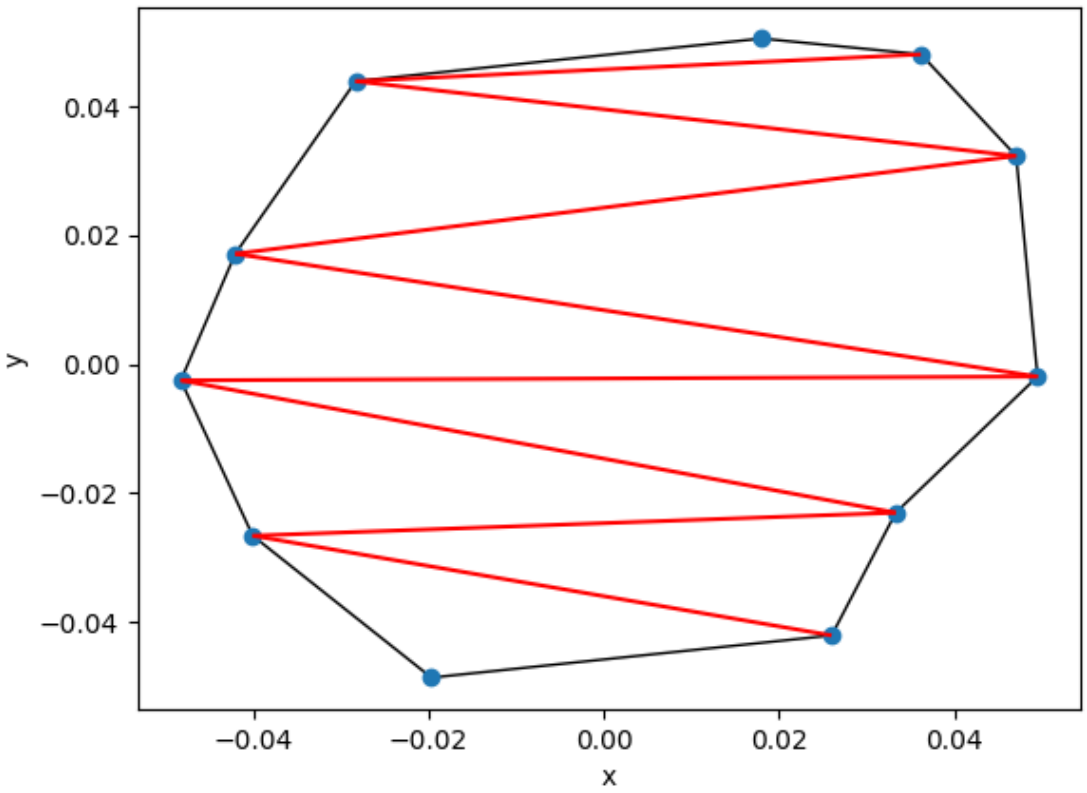
In [169…
```
t3 = [(0.018013488400366998, 0.05054348863048477), (-0.02833731805124591,
```

In [170…
```
assert is_y_monotonic(t3), 'Polygon is not y-monotonic'
tri = triangulation(t3)
draw_polygon_tri(t3,[[t3[diag[0]], t3[diag[1]]] for diag in tri])
```

Figure



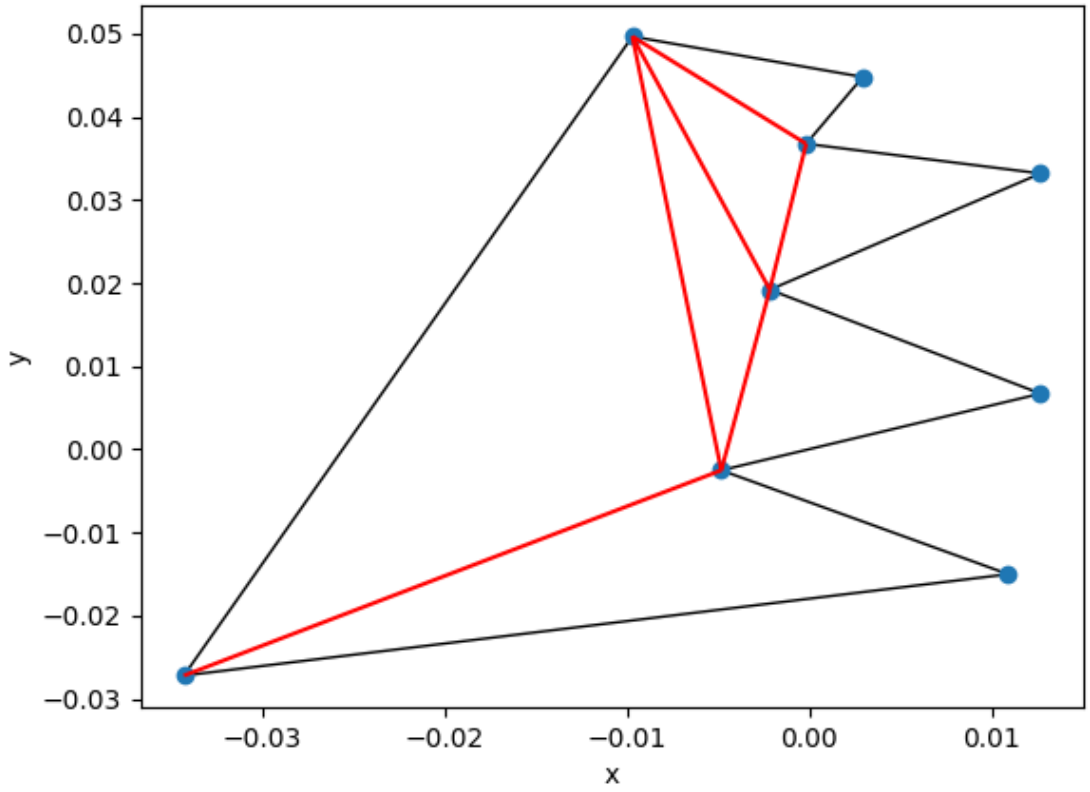
Figure



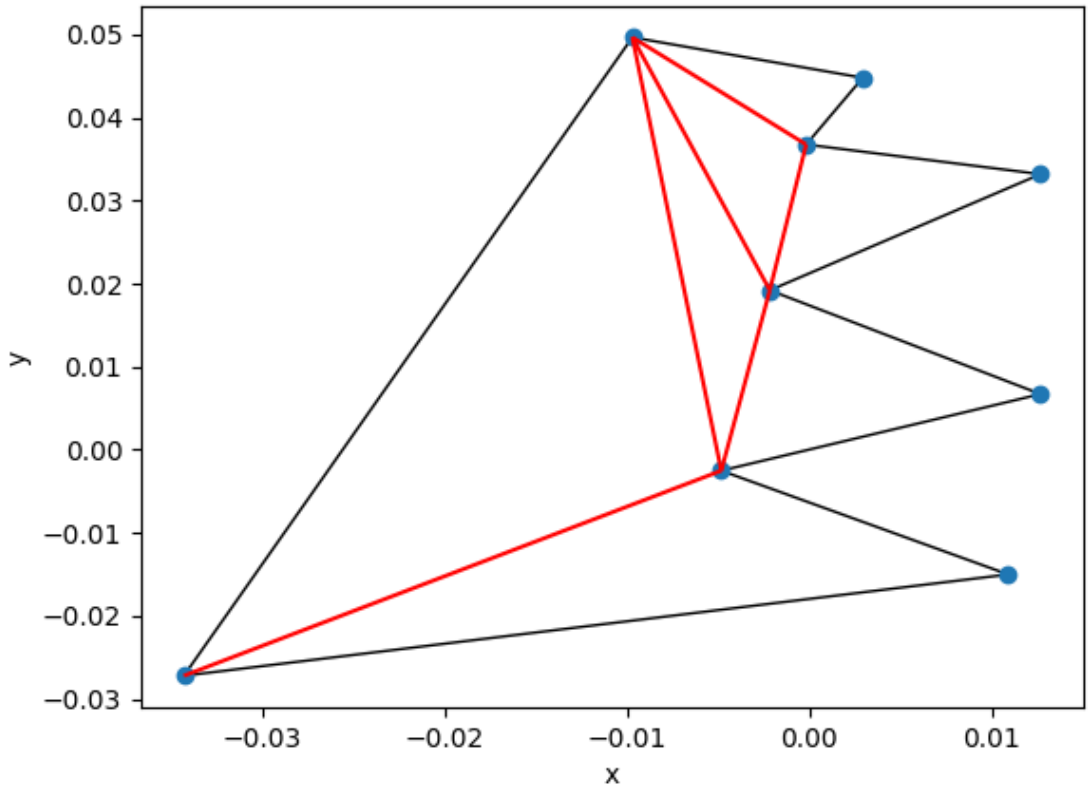
In [171…
```
t5 = [(-0.009708285793181393, 0.049673460634239885), (-0.0343252212770523
```

In [172…
```
assert is_y_monotonic(t5), 'Polygon is not y-monotonic'
tri = triangulation(t5)
draw_polygon_tri(t5,[[t5[diag[0]], t5[diag[1]]] for diag in tri])
```

Figure



Figure

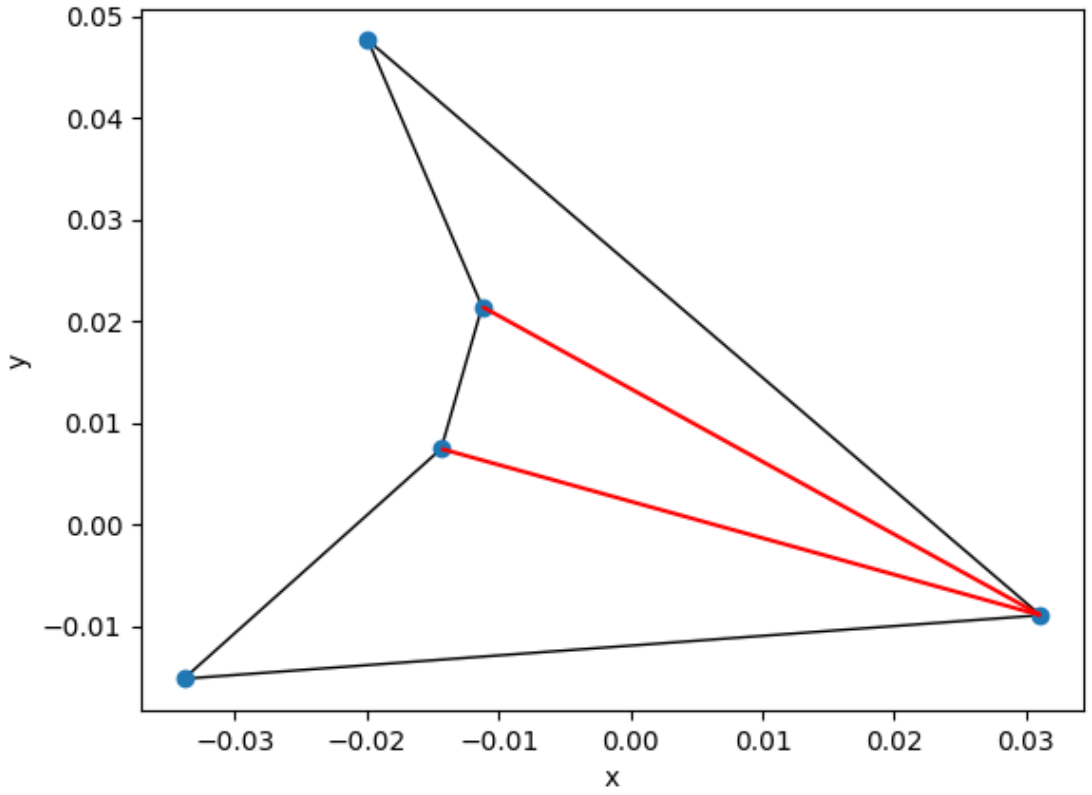


In [162…
```
t6 = [(-0.019909898696407198, 0.047644240371076646), (-0.0112607051480200
```

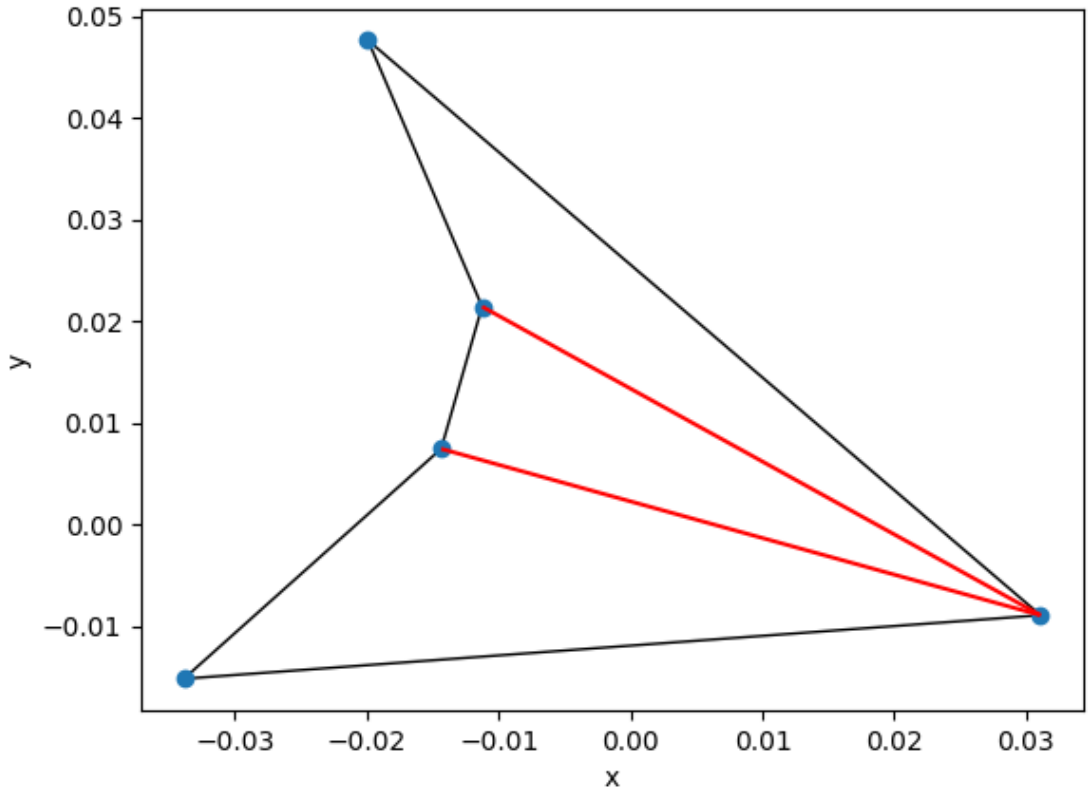
In [163…

```
assert is_y_monotonic(t6), 'Polygon is not y-monotonic'
tri = triangulation(t6)
draw_polygon_tri(t6,[[t6[diag[0]], t6[diag[1]]] for diag in tri])
```

Figure



Figure



In [ ]:

In [ ]:

In [ ]: