# Projekt: Otoczka wypukła zbioru czterech punktów

# Skład zespołu nr 34

Norbert Żuchowski,  
Nr albumu: 178046

* Implementacja funkcji matematycznych:
* cross(o, a, b) – obliczanie orientacji i iloczynu wektorowego
* is\_point\_on\_segment(p, a, b) – sprawdzanie przynależności punktu do odcinka
* Realizacja algorytmu monotone chain w funkcji convex\_hull(points)

*……………………*

Filip Radziun

Nr albumu: 178790

* Parsowanie i walidacja danych wejściowych w parse\_input\_points(text)
* Obsługa wyjątków i komunikatów błędów (ValueError, ostrzeżenia Tkinter)

*……………………*

Michał Botor

Nr albumu: 177995

* Budowa interfejsu GUI w Tkinter (root, Text, Button)
* Funkcja compute\_and\_plot() – koordynacja wczytywania, walidacji, wywołań algorytmu i wizualizacji

*……………………*

Mateusz Dąbrowski

Nr albumu: 178934

* Wizualizacja wyników funkcją plot\_points\_and\_hull(points, hull) z wykorzystaniem matplotlib
* Zarządzanie katalogiem i unikalnymi nazwami plików (get\_output\_directory, get\_unique\_filename)
* Przygotowanie pełnej dokumentacji projektu (pseudokod, schematy blokowe, opis funkcji)

*……………………*

# Manual

Plik otoczkaWypukla-v4.exe jest wersją kodu pliku otoczkaWypukla-v4.py, plik ten został zbudowany za pomocą narzędzia PyInstaller. Kod i działanie programu obu plików jest jednakowe.

Aby uruchomić program należy kliknąć plik otoczkaWypukla-v4.exe i wybrać opcje uruchom.

## Zadanie, które program ma realizować

## Program wyznacza otoczkę wypukłą czterech punktów na płaszczyźnie:

## • informuje, jakim zborem jest otoczka wypukła (czworokąt, trójkąt, odcinek, punkt),

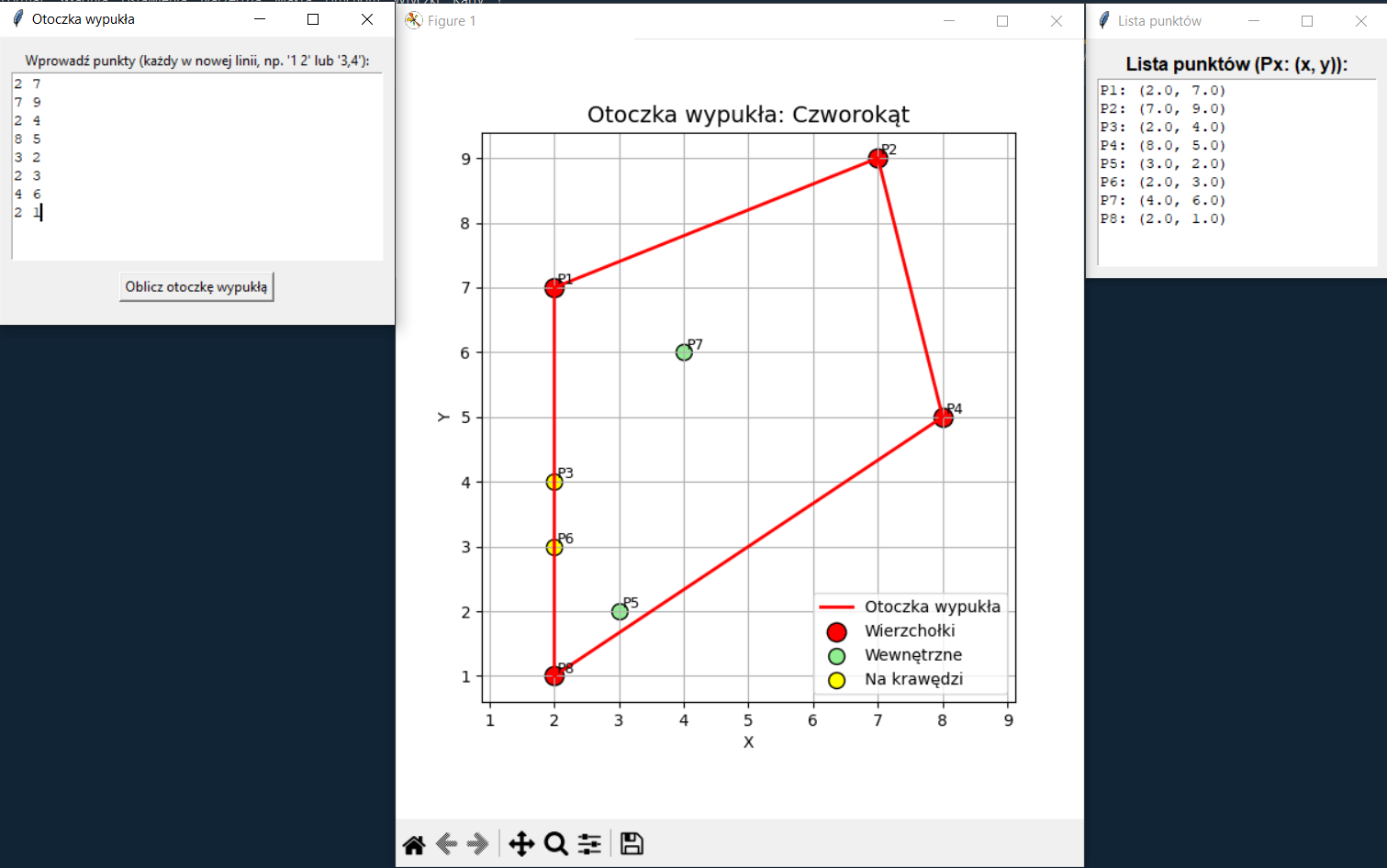
## • program wypisuje współrzędne kolejnych wierzchołków otoczki wypukłej.

## Dane wejściowe: współrzędne czterech punktów na płaszczyźnie.

## Niezgodności z założeniami przekazanymi w treści zadania

Program został uzupełniony o funkcje:

* jakim zborem jest otoczka wypukła
* wypisuje współrzędne kolejnych wierzchołków otoczki wypukłej



# Opis kodu

## Lista plików z kodem źródłowym wchodzących w skład programu

Main.py  
otoczkaWypukla-v4.exe

import tkinter as tk

from tkinter import messagebox

import matplotlib.pyplot as plt

import os

import sys

import threading

from PIL import Image, ImageDraw, ImageFont

def cross(o, a, b):

return (a[0] - o[0]) \* (b[1] - o[1]) - (a[1] - o[1]) \* (b[0] - o[0])

def is\_point\_on\_segment(p, a, b, eps=1e-9):

cross\_val = cross(a, b, p)

if abs(cross\_val) > eps:

return False

min\_x, max\_x = sorted([a[0], b[0]])

min\_y, max\_y = sorted([a[1], b[1]])

return min\_x - eps <= p[0] <= max\_x + eps and min\_y - eps <= p[1] <= max\_y + eps

def convex\_hull(points):

points = sorted(set(points))

if len(points) <= 1:

return points

lower = []

for p in points:

while len(lower) >= 2 and cross(lower[-2], lower[-1], p) <= 0:

lower.pop()

lower.append(p)

upper = []

for p in reversed(points):

while len(upper) >= 2 and cross(upper[-2], upper[-1], p) <= 0:

upper.pop()

upper.append(p)

return lower[:-1] + upper[:-1]

def get\_output\_directory(folder\_name="wykresy"):

if getattr(sys, 'frozen', False):

base\_dir = os.path.dirname(sys.executable)

else:

base\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

output\_dir = os.path.join(base\_dir, folder\_name)

if not os.path.exists(output\_dir):

os.makedirs(output\_dir)

return output\_dir

def get\_unique\_filename(base\_name="convex\_hull", ext="png"):

output\_dir = get\_output\_directory()

i = 1

while True:

filename = os.path.join(output\_dir, f"{base\_name}\_{i}.{ext}")

if not os.path.exists(filename):

return filename

i += 1

def save\_points\_as\_image(points):

output\_dir = get\_output\_directory()

filename = get\_unique\_filename(base\_name="lista\_punktow", ext="png")

width = 400

line\_height = 30

height = line\_height \* (len(points) + 2)

image = Image.new("RGB", (width, height), color="white")

draw = ImageDraw.Draw(image)

try:

font = ImageFont.truetype("arial.ttf", 16)

except:

font = ImageFont.load\_default()

draw.text((10, 10), "Lista punktów:", font=font, fill="black")

for i, (x, y) in enumerate(points):

draw.text((10, 10 + (i + 1) \* line\_height), f"P{i+1}: ({x}, {y})", font=font, fill="black")

image.save(filename)

print(f"Lista punktów zapisana do pliku: {filename}")

def plot\_points\_and\_hull(points, hull):

def plotting():

plt.figure(figsize=(6, 6.8))

hull\_set = set(hull)

on\_edge = []

inner = []

for pt in points:

if pt in hull\_set:

continue

for i in range(len(hull)):

a = hull[i]

b = hull[(i + 1) % len(hull)]

if is\_point\_on\_segment(pt, a, b):

on\_edge.append(pt)

break

else:

inner.append(pt)

if len(hull) > 1:

hx, hy = zip(\*(hull + [hull[0]]))

plt.plot(hx, hy, 'r-', linewidth=2, label='Otoczka wypukła')

if hull:

hx, hy = zip(\*hull)

plt.scatter(hx, hy, color='red', s=140, edgecolor='black', label='Wierzchołki')

if inner:

ix, iy = zip(\*inner)

plt.scatter(ix, iy, color='lightgreen', s=100, edgecolor='black', label='Wewnętrzne')

if on\_edge:

ex, ey = zip(\*on\_edge)

plt.scatter(ex, ey, color='yellow', s=100, edgecolor='black', label='Na krawędzi')

for i, (x, y) in enumerate(points):

plt.text(x + 0.05, y + 0.05, f'P{i+1}', fontsize=9)

typy\_otoczki = {1: "Punkt", 2: "Odcinek", 3: "Trójkąt", 4: "Czworokąt", 5: "Pięciokąt"}

n = len(hull)

typ = typy\_otoczki.get(n, f"{n}-kąt")

plt.title(f"Otoczka wypukła: {typ}", fontsize=14)

plt.subplots\_adjust(bottom=0.15)

plt.xlabel("X")

plt.ylabel("Y")

plt.axis('equal')

plt.legend()

plt.grid(True)

filename = get\_unique\_filename()

plt.savefig(filename, bbox\_inches='tight')

print(f"Wizualizacja została zapisana do pliku: {filename}")

plt.show()

threading.Thread(target=plotting).start()

def show\_points\_in\_window(points):

save\_points\_as\_image(points)

top = tk.Toplevel()

top.title("Lista punktów")

frame = tk.Frame(top, padx=10, pady=10)

frame.pack()

label = tk.Label(frame, text="Lista punktów (Px: (x, y)):", font=('Arial', 12, 'bold'))

label.pack()

text\_box = tk.Text(frame, width=30, height=min(25, len(points)+2), font=("Courier New", 10))

text\_box.pack()

for i, (x, y) in enumerate(points):

text\_box.insert(tk.END, f"P{i+1}: ({x}, {y})\n")

text\_box.config(state=tk.DISABLED)

def parse\_input\_points(text):

lines = text.strip().splitlines()

points = []

for line in lines:

parts = line.replace(',', ' ').split()

if len(parts) != 2:

raise ValueError(f"Nieprawidłowy format: '{line}'")

x, y = float(parts[0]), float(parts[1])

points.append((x, y))

return points

def compute\_and\_plot():

try:

points = parse\_input\_points(text\_input.get("1.0", tk.END))

if len(points) < 3:

messagebox.showwarning("Za mało punktów", "Wprowadź co najmniej 3 punkty.")

return

hull = convex\_hull(points)

show\_points\_in\_window(points)

plot\_points\_and\_hull(points, hull)

except Exception as e:

messagebox.showerror("Błąd", str(e))

# GUI

root = tk.Tk()

root.title("Otoczka wypukła")

frame = tk.Frame(root, padx=10, pady=10)

frame.pack()

label = tk.Label(frame, text="Wprowadź punkty (każdy w nowej linii, np. '1 2' lub '3,4'):")

label.pack()

text\_input = tk.Text(frame, width=40, height=10)

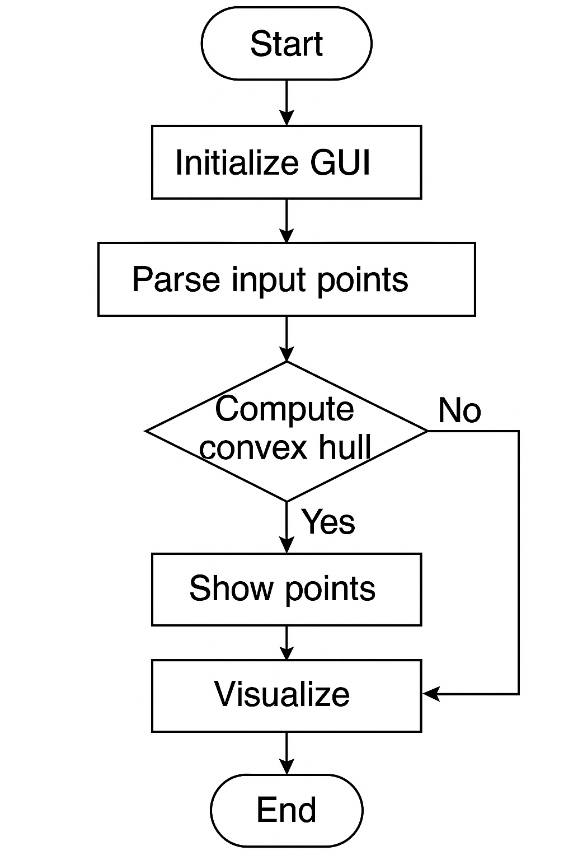
text\_input.pack()

button = tk.Button(frame, text="Oblicz otoczkę wypukłą", command=compute\_and\_plot)

button.pack(pady=10)

root.mainloop()

## Schemat algorytmu i pseudokod (z odniesieniami do kodu programu)



Oto uproszczony pseudokod aplikacji wyznaczającej otoczkę wypukłą z krótkimi komentarzami:

PSEUDOKOD – Otoczka wypukła (z GUI)  
Start

1. Inicjalizacja GUI

* Utwórz główne okno aplikacji z tytułem "Otoczka wypukła".
* Dodaj pole tekstowe do wprowadzania punktów (w formacie "x y" lub "x,y", po jednym na linię).
* Dodaj przycisk "Oblicz otoczkę wypukłą".

2. Po kliknięciu przycisku

* Odczytaj dane wejściowe z pola tekstowego.
* Przetwórz każdą linię jako punkt w postaci współrzędnych (x, y).
* Jeśli format punktu jest niepoprawny, pokaż komunikat o błędzie.
* Sprawdź, czy podano co najmniej 3 punkty:
* Jeśli nie, pokaż ostrzeżenie.

3. Oblicz otoczkę wypukłą

* Usuń duplikaty punktów i posortuj je.
* Zastosuj algorytm Grahama lub Andrew’s monotone chain do wyznaczenia otoczki wypukłej:
* Zbuduj dolną i górną część otoczki.
* Połącz je w jedną listę punktów tworzących otoczkę wypukłą.

4. Wyświetl listę punktów

* Utwórz nowe okno z listą punktów w formacie P1: (x, y), P2: (x, y), itd.
* Zapisz tę listę jako obraz PNG z tekstową reprezentacją punktów.

5. Wizualizacja otoczki

* Otwórz nowy wątek, aby nie blokować GUI.
* Utwórz wykres:
* Narysuj otoczkę wypukłą jako czerwoną linię.
* Wierzchołki otoczki: czerwone kropki.
* Punkty leżące na krawędziach otoczki: żółte kropki.
* Punkty wewnętrzne: zielone kropki.
* Podpisz każdy punkt etykietą P1, P2, itd.
* Dodaj tytuł np. "Otoczka wypukła: Trójkąt" (w zależności od liczby wierzchołków).
* Zapisz wykres jako obraz PNG.
* Wyświetl wykres użytkownikowi.

6. Funkcje pomocnicze

* cross(o, a, b): Oblicza iloczyn wektorowy.
* is\_point\_on\_segment(p, a, b): Sprawdza, czy punkt p leży na odcinku ab.
* get\_output\_directory(): Tworzy i zwraca ścieżkę do folderu wykresy.
* get\_unique\_filename(base\_name, ext): Generuje unikalną nazwę pliku w folderze wynikowym.
* save\_points\_as\_image(points): Zapisuje punkty jako obraz z tekstem.

Koniec

## Zadanie, które program ma realizować

Program wczytuje współrzędne dowolnej liczby punktów i oblicza ich otoczkę wypukłą, prezentując wynik w formie wykresu.

Pole tekstowe do wprowadzenia punktów; przycisk „Oblicz otoczkę wypukłą”; obsługa błędów formatu i liczby punktów.

- main.py – główny skrypt z funkcjami matematycznymi (cross, is\_point\_on\_segment, convex\_hull), parsowaniem danych, rysowaniem wykresu i GUI Tkinter.

- Pseudokod algorytmu monotone chain: sortowanie punktów; budowa dolnej i górnej otoczki; łączenie list wierzchołków.