# WSI - ćwiczenie 1.
# Algorytmy ewolucyjne i genetyczne

13 października 2025

## 1 Sprawy organizacyjne

1. Ćwiczenie realizowwane jest samodzielnie.
2. Ćwiczenie wykonywane jest w języku Python.
3. Ćwiczenie powinno zostać oddane najpóźniej do 27.10.2025. W ramach oddawania ćwiczenia należy zademonstrować prowadzącemu działanie kodu oraz utworzyć pull request (z kodem oraz raportem) który prowadzący będzie mógł komentować.
4. Kod oraz raport powinien być umieszczony na repozytorium `gitlab-stud.elka.pw.edu.pl`, a pull request utworzony do konta prowadzącego: **@djagodzi** (dla nazwisk A-N) i **@pzawist2** (dla nazwisk O-Z)
5. Implementacja powinna być zgodna z interfejsem dostępnym w repozytorium `https://gitlab-stud.elka.pw.edu.pl/jlyskawa/wsi-template`
6. Raport powinien być w postaci pliku .pdf, .html albo być częścią notebooka jupyterowego. Powinien zawierać opis eksperymentów, uzyskane wyniki wraz z komentarzem oraz wnioski.
7. Na ocenę wpływa poprawność oraz jakość kodu i raportu.
8. Można korzystać z pakietów do obliczeń numerycznych, takich jak *numpy*
9. Implementacja powininna być ogólna - w szczególności działać dla dowolnej funkcji $\{0,1\}^n \to \mathbf{R}$

## 2 Ćwiczenie

Celem ćwiczenia jest implementacja algorytmu genetycznego z mutacją, selekcją ruletkową, krzyżowaniem jednopunktowym oraz sukcesją generacyjną.

Następnie należy zbadać działanie algorytmu na przykładzie problemu opisanego w rozdziale 3.

- Zastanów się, jakie mogło by być trywialne rozwiązanie. Jaki byłby jego wynik? Potraktuj to rozwiązanie jako punkt odniesienia w eksperymentach.
- Jakie jest górne ograniczenie wyniku?
- Zmierz, jaki wynik osiąga rozwiązanie losowe.

Należy znaleźć zestaw hiperparametrów który daje dobry wynik (wartość funkcji celu powyżej -4), a następnie znaleźć zbadać wpływ wybranego przez siebie hiperparametru.

**Dla chętnych** Uwzględnij ograniczenie, że silniki mogą być włączone przez, w sumie, maksymalnie 300 sekund (wektor osobnika może zawierać maksymalnie 300 bitów o wartości 1).

*Ta część zadania nie wpływa bezpośrednio na wynik.*

# 3 Opis problemu

Zadaniem jest optymalizacja sterowania obiektem poruszającym się w płaszczyźnie pionowej i poziomej przez maksymalnie 20 sekund tak, aby wylądował 350 metrów od startu.

W każdym kroku czasowym trwającym 0.1s silnik pionowy oraz silnik poziomy mogą być, niezależnie od siebie, włączone albo wyłaczone. Włączony silnik zapewnia przyspieszenie $15\frac{m}{s^2}$. Opór powietrza w każdej osi wynosi $0.5v\frac{m}{s^2}$. Przyspieszenie grawitacyjne to $9.8\frac{m}{s^2}$.

Funkcja celu dla problemu maksymalizacji to $-d^2$, gdzie $d$ to różnica między odległością przebytą a zadaną (350 metrów).

# 4 Kod ewaluacji osobnika (z użyciem numpy)

Poniższa implementacja zadziała zarówno dla pojedynczego osobnika jak i dla kolekcji osobników.

```
import numpy as np


def calc_path(control: np.ndarray, time: int = 200, dt: float = 0.1):
    pathx = []
    pathz = []

    posx = np.zeros(control.shape[:-1])
    posz = np.zeros(control.shape[:-1])
    velx = np.zeros(control.shape[:-1])
    velz = np.zeros(control.shape[:-1])
    control = control.reshape(*control.shape[:-1], time, 2)
    t = 0
    pathx.append(posx)
    pathz.append(posz)
    while (posz >= 0).any():
        if t < time:
            cx = control[..., t, 0]
            cz = control[..., t, 1]
        else:
            cx = 0
            cz = 0
        velx = velx + (cx * 15 - 0.5 * velx) * dt
        velz = velz + (cz * 15 - 9.8 - 0.5 * velz) * dt

        velx = velx * (posz >= 0)
        velz = velz * (posz >= 0)

        posx = posx + velx * dt
        posz = posz + velz * dt

        pathx.append(posx)
        pathz.append(posz)
        t += 1
    return pathx, pathz


def calc_target(control: np.ndarray):
    pathx, pathz = calc_path(control)
    return -(pathx[-1] - 350) ** 2
```