Sprawozdanie z MIO laboratorium 08 - Marcin Knapczyk

Zadanie 1

Proszę zaproponować swoją implementację algorytmu genetycznego w celu znalezienia maksimum funkcji

$$f(x) = \cos(80x + 0.3) + 3x^{-0.9} - 2$$

w przedziale $[0.01, 1]$. Dla $x = 0$ proszę przyjąć $f(x) = 0$ (choć nie powinno być używane w zadaniu). Proszę porównać działanie algorytmu:

- Dla kodowania w naturalnym kodzie binarnym i w kodzie Graya
- Dla szansy mutacji wynoszącej 0, 0.1, 0.5 i 1.0
- Dla selekcji ruletkowej i dla selekcji progowej. W selekcji progowej dzielimy populację na dwie grupy: na $\gamma\%$ najlepszych i na pozostałych. Osobniki w grupie $\gamma\%$ najlepszych mają równą szansę na reprodukcję, pozostałe mają zerową szansę na reprodukcję. Proszę sprawdzić wyniki dla $\gamma = 30$ i $\gamma = 60$.

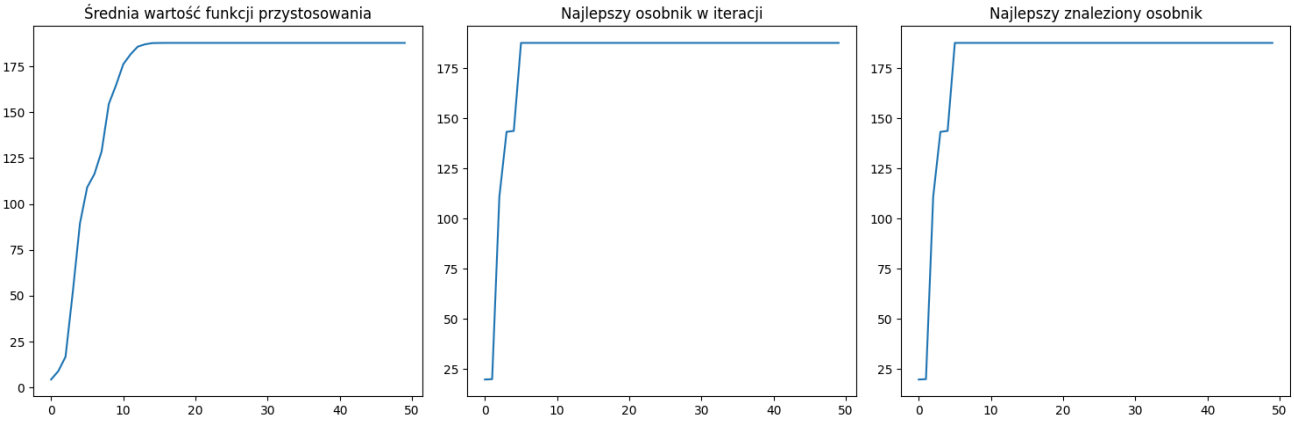
Za każdym razem proszę podać średnie wyniki dla 10 wywołań algorytmu i przedstawić przykładowe przebiegi algorytmu na wykresach (dla jednego z wywołań).

```
import math
import random
import matplotlib.pyplot as plt
import numpy as np


# zadana funkcja
def f(x):
    return np.cos(80 * x + 0.3) + 3 * (x**(-0.9)) - 2 if x != 0 else 0

def binary_to_gray(bits):
    gray = [bits[0]]
    for i in range(1, len(bits)):
        gray.append(bits[i-1] ^ bits[i])
    return gray

def gray_to_binary(gray):
    bits = [gray[0]]
    for i in range(1, len(gray)):
        bits.append(bits[i-1] ^ gray[i])
    return bits

# klasa reprezentująca osobnika
class Solution:
    def __init__(self, coding="binary", randomize_genes=False):
        self.genes = [0] * 16
        self.coding = coding
        if randomize_genes:
            for i in range(16):
                self.genes[i] = random.randint(0, 1)

    def decode(self):
        if self.coding == "gray":
            bits = gray_to_binary(self.genes)
        else:
            bits = self.genes
        number = int("".join(str(x) for x in bits), 2)
        return 0.01 + (number /  (2**len(bits) - 1)) * (1 - 0.01)

    def get_adaptation(self):
        x = self.decode()
        return f(x)
```

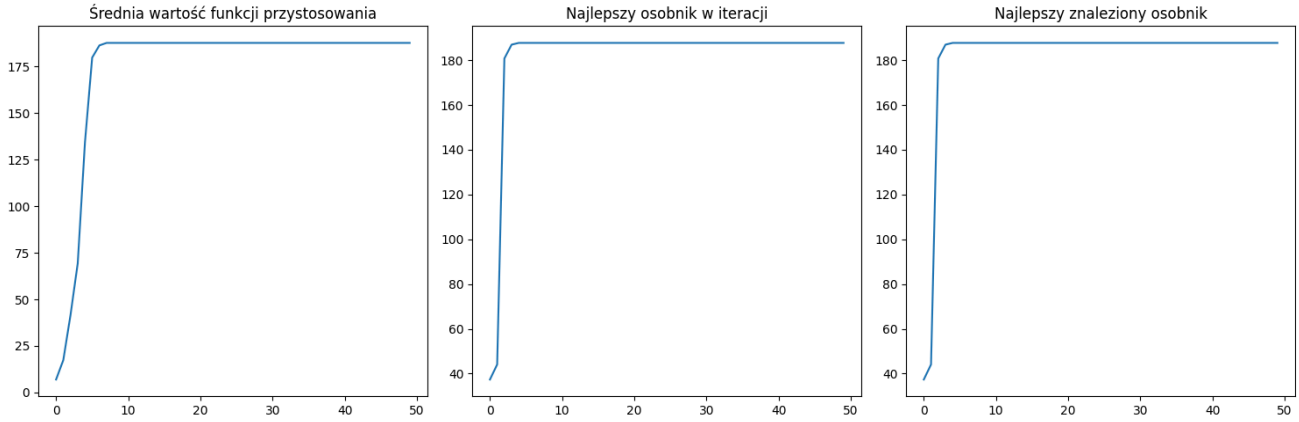
```
    def one_point_crossover(self, other_solution):
        cut_position = random.randint(0, 15)
        new_solution = Solution(coding=self.coding)
        new_solution.genes = self.genes[:cut_position] + other_solution.genes[cut_position:]
        return new_solution

    def mutation(self):
        mutation_position = random.randint(0, 15)
        self.genes[mutation_position] ^= 1  # obrót bitu


def run_genetic_algorithm(coding="binary", mutation_chance=0.1, selection_method="roulette",
    gamma=30, population_size=50, iterations=50
):
    population = [Solution(coding=coding, randomize_genes=True) for _ in range(population_size)]

    best_solution = None
    best_solution_adaptation = float('-inf')
    best_iteration_found = 0

    avgs = []
    bests_local = []
    bests_global = []

    for iteration in range(iterations):
        adaptations = [p.get_adaptation() for p in population]

        # najlepszy lokalnie
        local_best_idx = np.argmax(adaptations)
        local_best_solution = population[local_best_idx]
        if adaptations[local_best_idx] > best_solution_adaptation:
            best_solution = local_best_solution
            best_solution_adaptation = adaptations[local_best_idx]
            best_iteration_found = iteration

        avgs.append(np.mean(adaptations))
        bests_local.append(np.max(adaptations))
        bests_global.append(best_solution_adaptation)

        if selection_method == "roulette":  # selekcja ruletkowa
            min_adapt = min(adaptations)
            max_adapt = max(adaptations)
            if max_adapt - min_adapt > 0:
                roulette_wheel = [(a - min_adapt) / (max_adapt - min_adapt) for a in adaptations]
            else:
                roulette_wheel = [1 for _ in adaptations]
            parents = [random.choices(population, weights=roulette_wheel, k=2) for _ in range(population_size)]

        else:  # selekcja progowa
            sorted_indices = np.argsort(adaptations)[::-1]
            num_elites = int(gamma/100 * population_size)
            elite_indices = sorted_indices[:num_elites]
            elite_population = [population[i] for i in elite_indices]
            parents = [random.choices(elite_population, k=2) for _ in range(population_size)]

        children = [p1.one_point_crossover(p2) for p1, p2 in parents]

        for child in children:
            if random.random() < mutation_chance:
                child.mutation()

        population = children

    return best_solution, best_solution_adaptation, best_iteration_found, avgs, bests_local, bests_global

def run_experiment(coding="binary", mutation_chances=[0.1], selection_methods=["roulette"], gammas=[30]):
    runs = 10
    for coding_type in [coding]:
        for mutation_chance in mutation_chances:
            for selection_method in selection_methods:
                gamma_values = gammas if selection_method == "threshold" else [None]
                for gamma in gamma_values:
                    results = []
                    best_solutions = []  # wszystkie najlepsze osobniki
                    print(f"\n\nCODING: {coding_type} | MUTATION: {mutation_chance} | SELECTION: {selection_method} | GAMMA: {gamma if
                    for run in range(runs):
                        best_solution, best_adaptation, found_iter, avgs, bests_local, bests_global = run_genetic_algorithm(
                            coding=coding_type,
                            mutation_chance=mutation_chance,
                            selection_method=selection_method,
                            gamma=gamma if gamma is not None else 30
                        )
                        results.append(best_adaptation)
                        best_solutions.append((best_solution, best_adaptation, found_iter, avgs, bests_local, bests_global))

                    print(f"Average best adaptation over {runs} runs: {np.mean(results):.5f}")

                    # najlepsze rozwiązanie
                    best_idx = np.argmax(results)
                    sample_solution, sample_adaptation, sample_iter, avgs, bests_local, bests_global = best_solutions[best_idx]

                    # najlepsze wyniki
                    print('---')
                    print('Best solution genes:', sample_solution.genes)
                    print('Decoded value (x):', sample_solution.decode())
                    print('Found in iteration:', sample_iter)
                    print('Largest function value found:', sample_adaptation)
```

```
                    print('True maximum function value: 187.740799465')

                    # wykresy przebiegu
                    plt.figure(figsize=(15, 5))
                    plt.subplot(1, 3, 1)
                    plt.plot(avgs)
                    plt.title('Średnia wartość funkcji przystosowania')
                    plt.subplot(1, 3, 2)
                    plt.plot(bests_local)
                    plt.title('Najlepszy osobnik w iteracji')
                    plt.subplot(1, 3, 3)
                    plt.plot(bests_global)
                    plt.title('Najlepszy znaleziony osobnik')
                    plt.tight_layout()
                    plt.show()

# eksperymenty
run_experiment(
    coding="binary",
    mutation_chances=[0.0, 0.1, 0.5, 1.0],
    selection_methods=["roulette", "threshold"],
    gammas=[30, 60]
)
run_experiment(
    coding="gray",
    mutation_chances=[0.0, 0.1, 0.5, 1.0],
    selection_methods=["roulette", "threshold"],
    gammas=[30, 60]
)
```

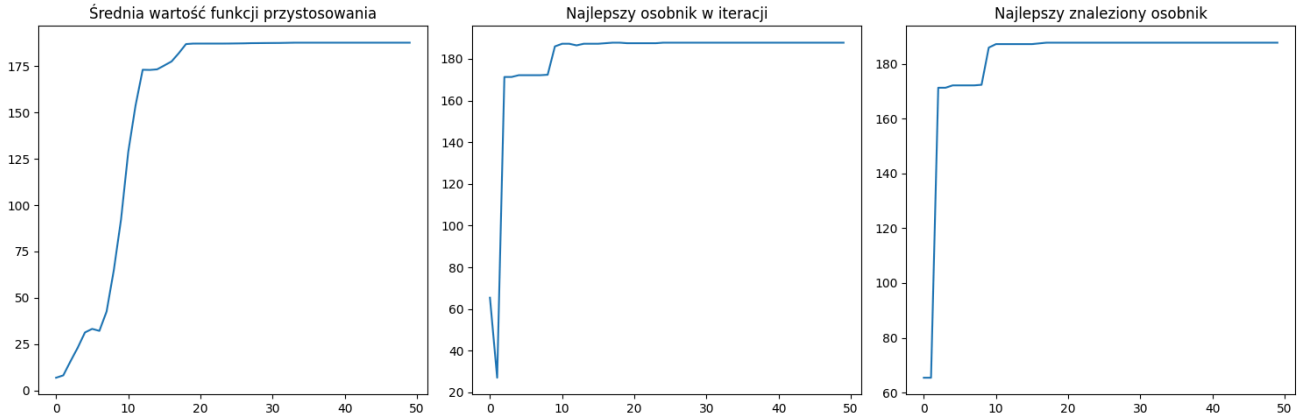
```
CODING: binary | MUTATION: 0.0 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 169.71610
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 5
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



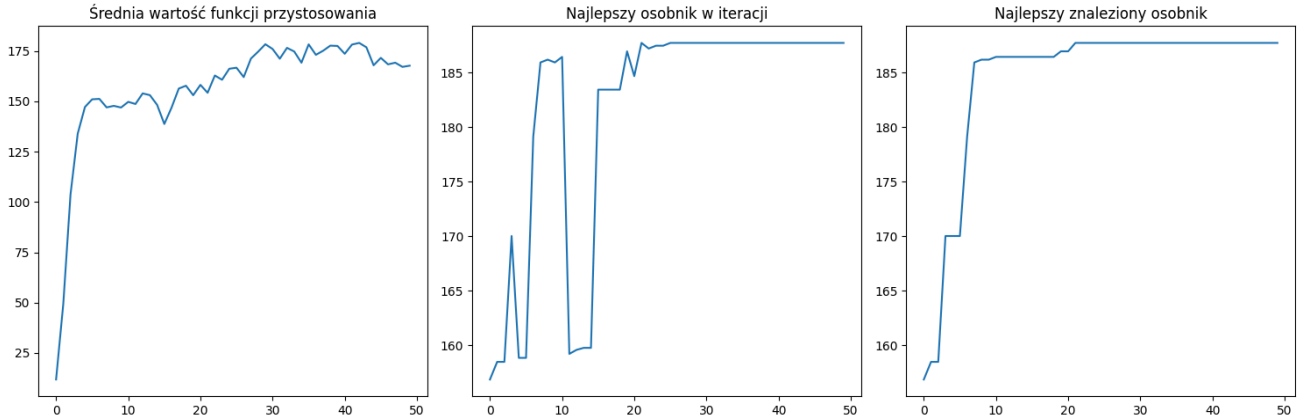

```
CODING: binary | MUTATION: 0.0 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 137.44217
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 4
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



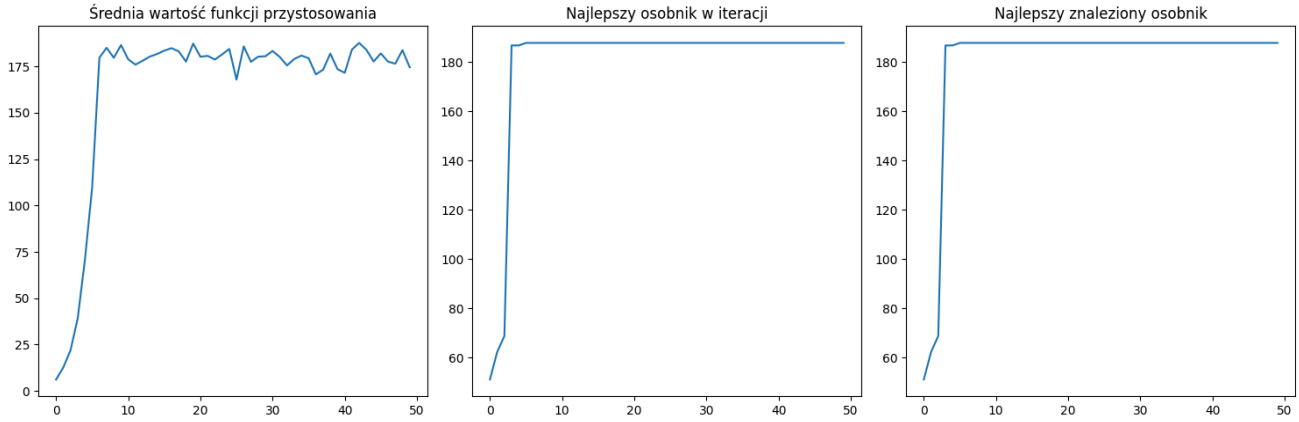

```
CODING: binary | MUTATION: 0.0 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 182.44666
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 17
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



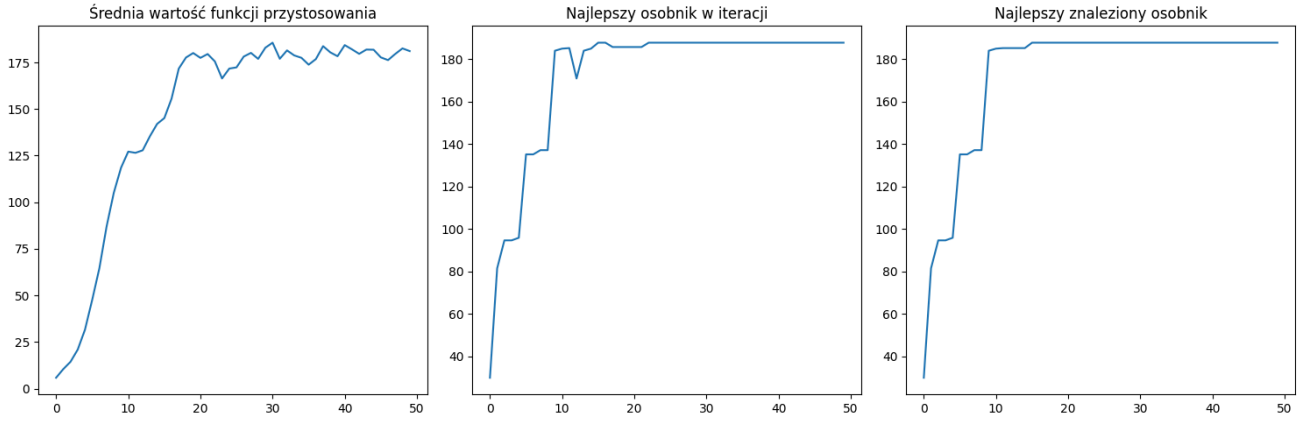

```
CODING: binary | MUTATION: 0.1 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 187.25920
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 21
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



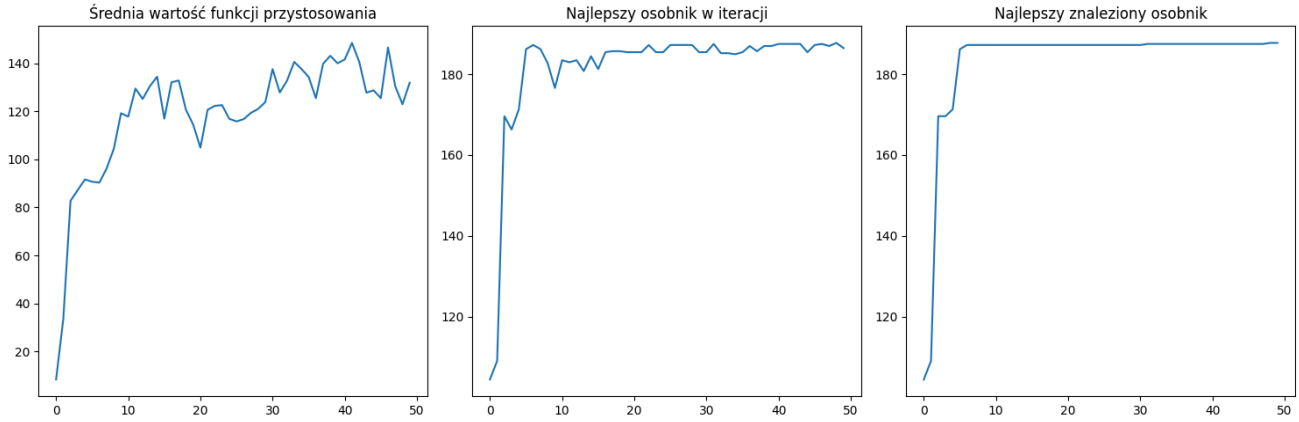
```
CODING: binary | MUTATION: 0.1 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 5
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



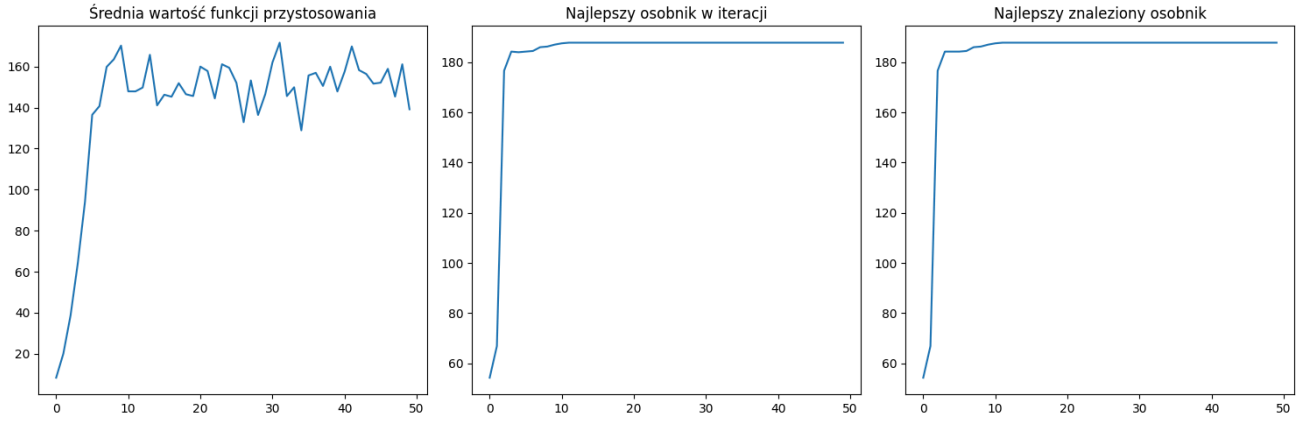

```
CODING: binary | MUTATION: 0.1 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 15
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



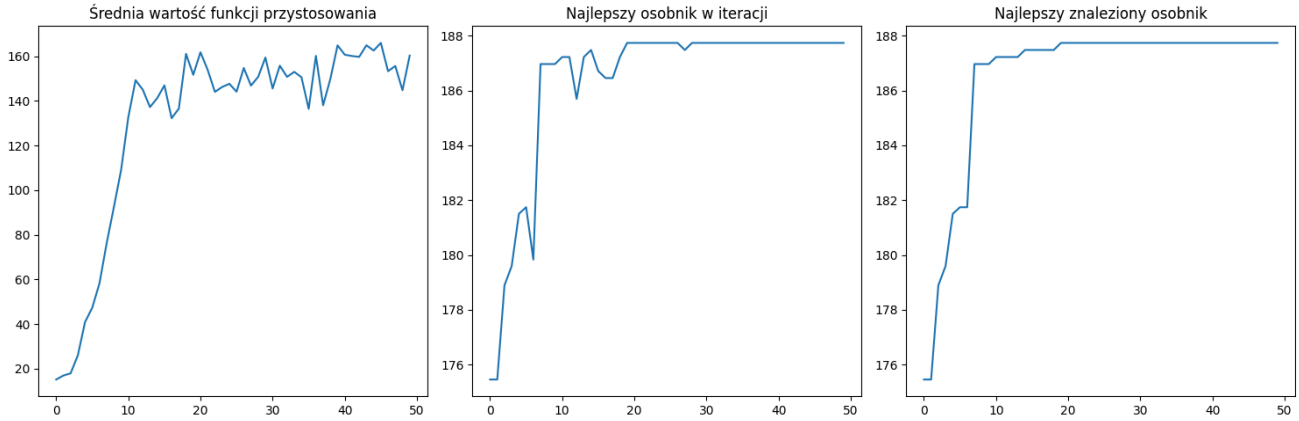

```
CODING: binary | MUTATION: 0.5 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 187.71499
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 48
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



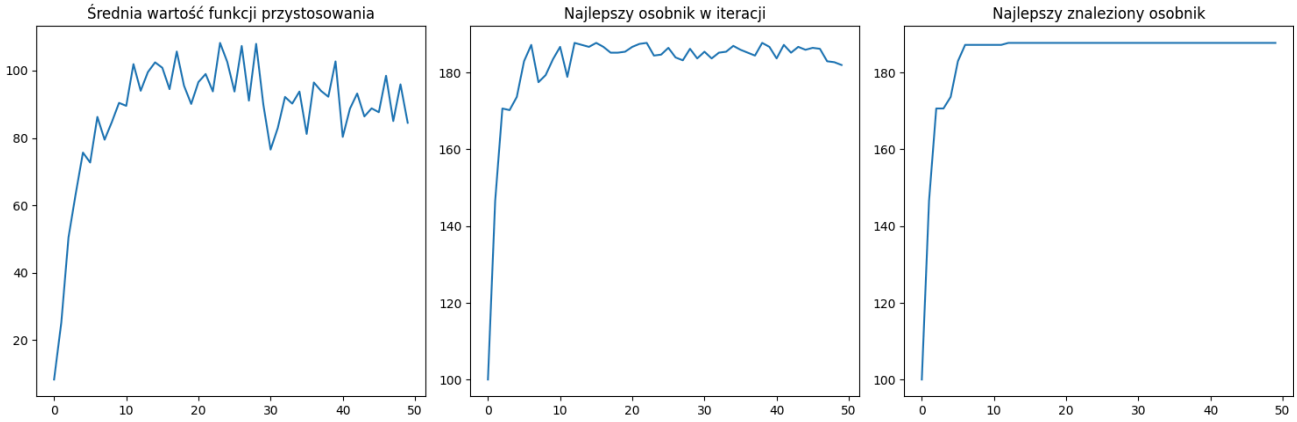

```
CODING: binary | MUTATION: 0.5 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 11
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



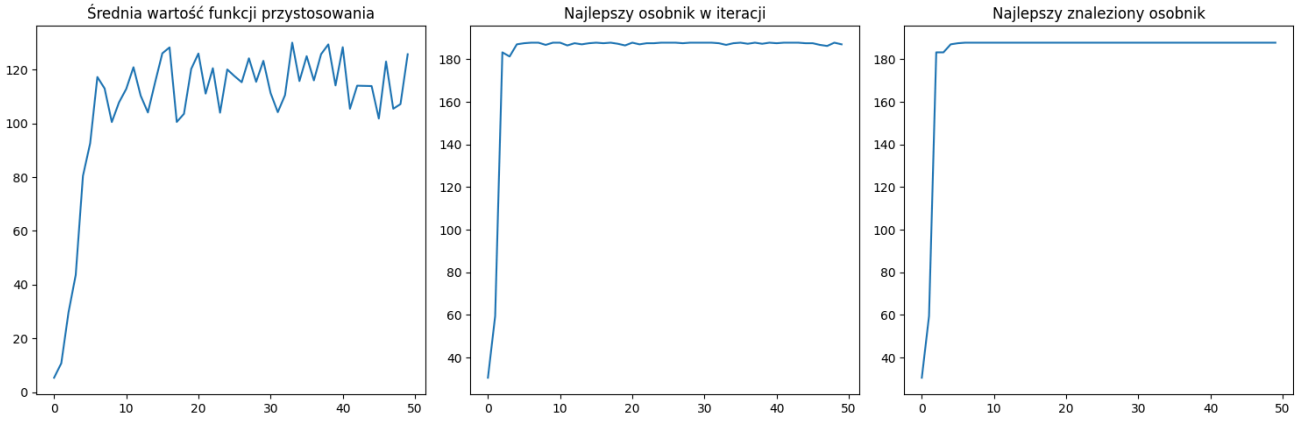

```
CODING: binary | MUTATION: 0.5 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 19
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



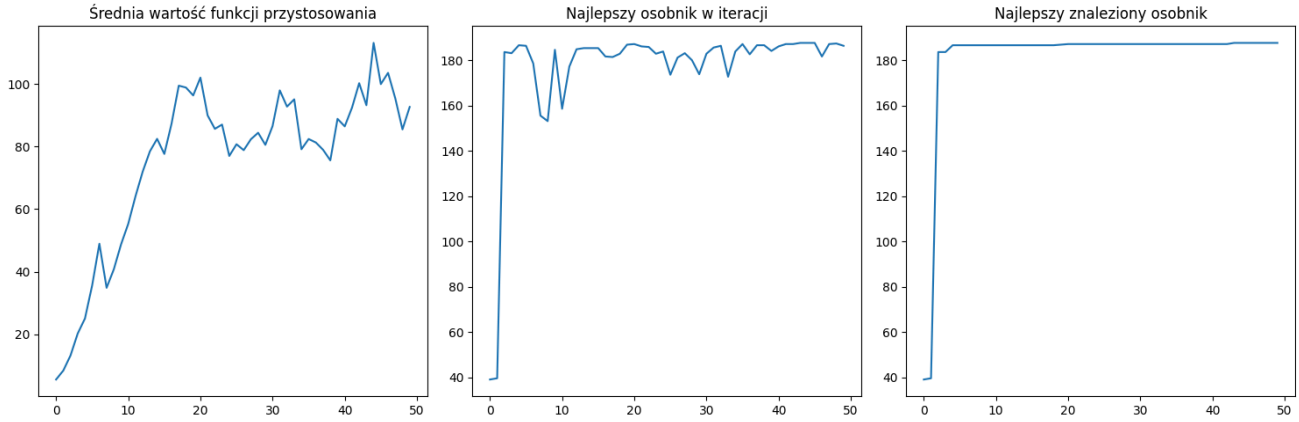
```
CODING: binary | MUTATION: 1.0 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 12
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



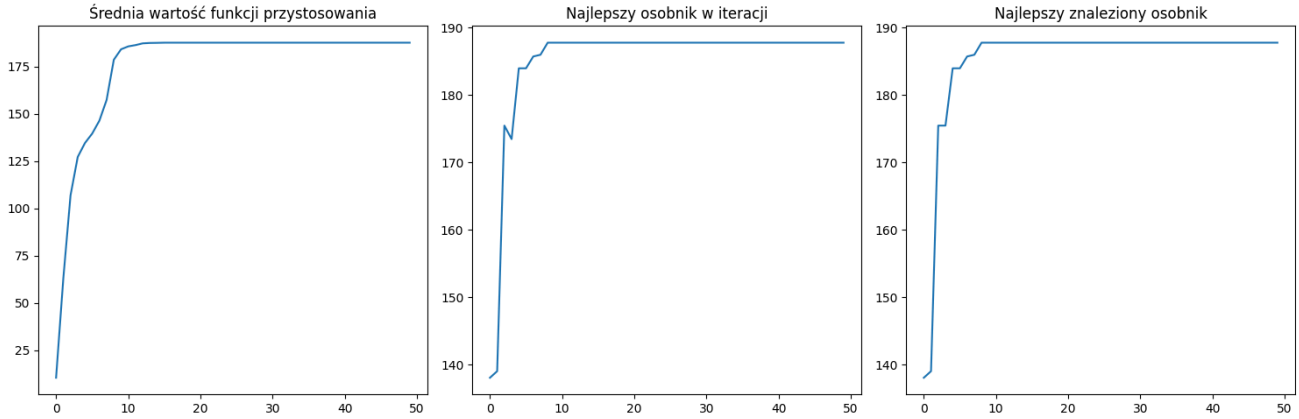

```
CODING: binary | MUTATION: 1.0 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 6
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



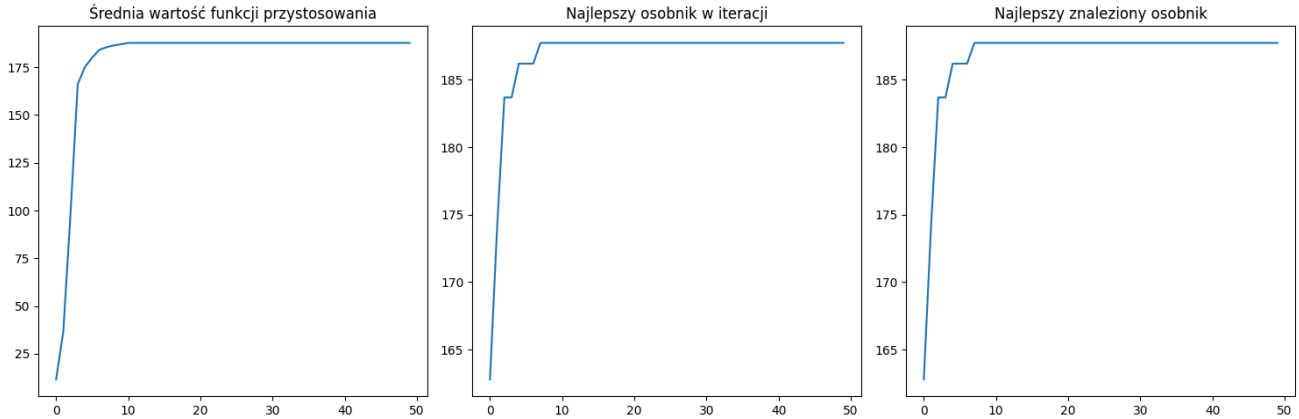

```
CODING: binary | MUTATION: 1.0 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.68919
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 43
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



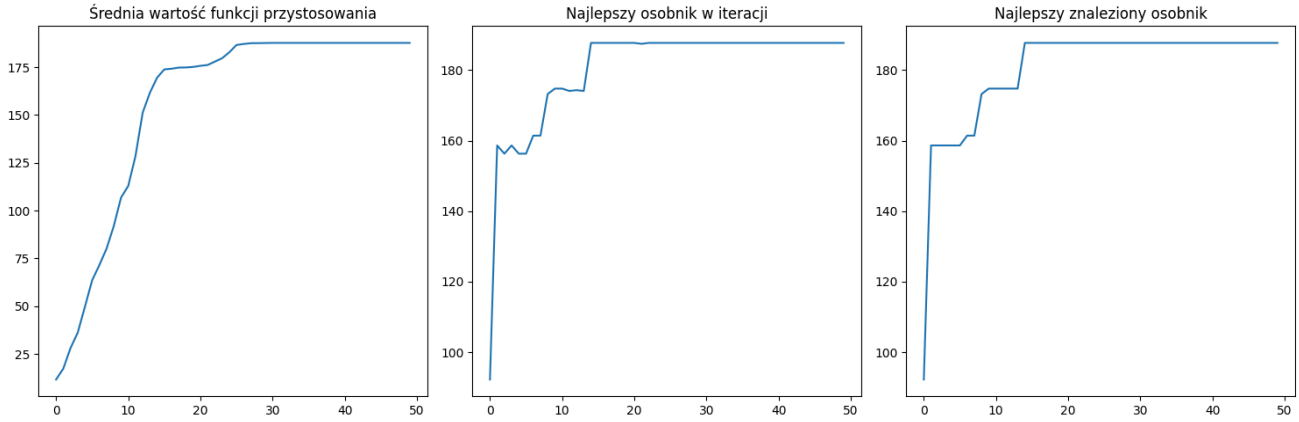

```
CODING: gray | MUTATION: 0.0 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 152.08070
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 8
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



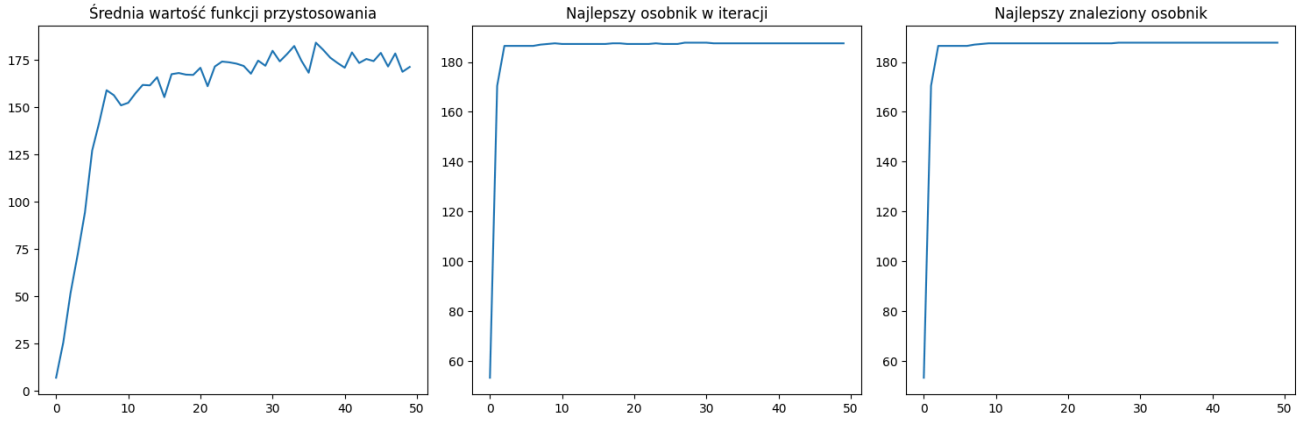

```
CODING: gray | MUTATION: 0.0 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 97.97408
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 7
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



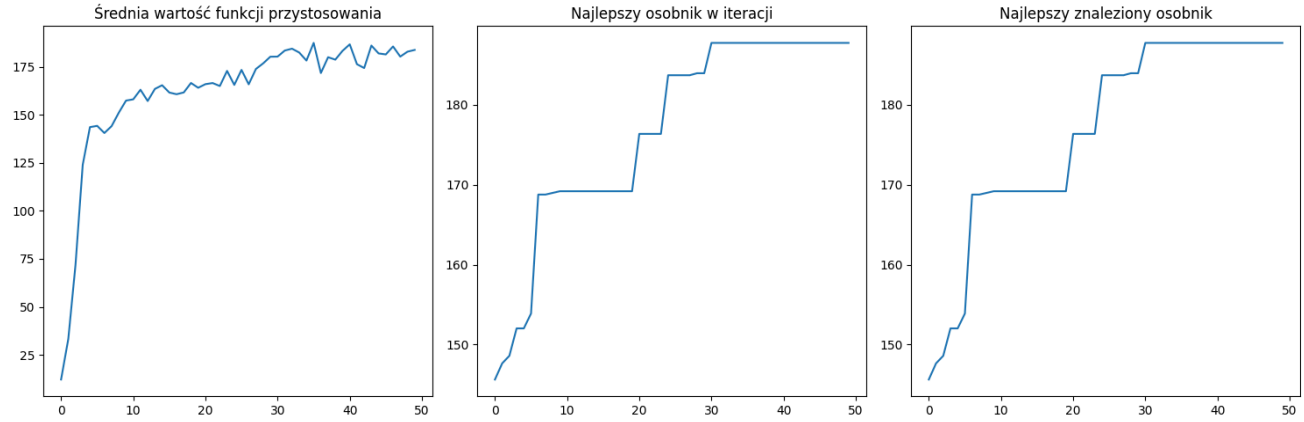
```
CODING: gray | MUTATION: 0.0 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 130.43946
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 14
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```




```
CODING: gray | MUTATION: 0.1 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 185.89637
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 27
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```

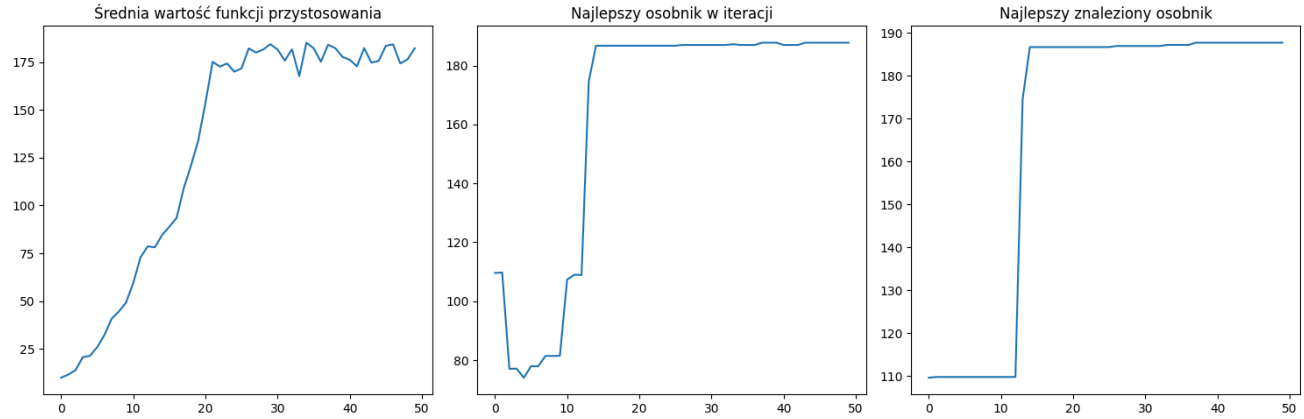



```
CODING: gray | MUTATION: 0.1 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.53639
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 30
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



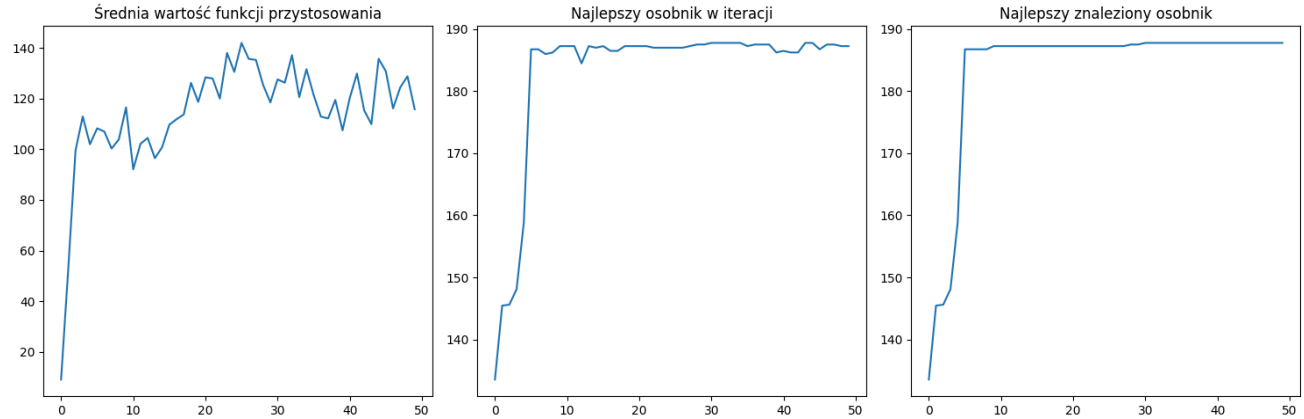
CODING: gray | MUTATION: 0.1 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.22883

Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 37
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465

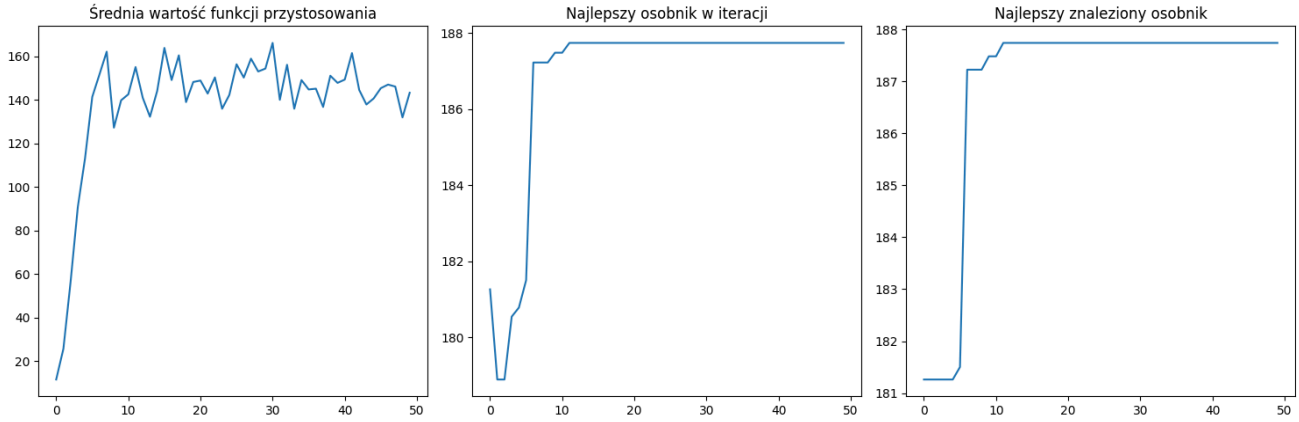


CODING: gray | MUTATION: 0.5 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 187.74080

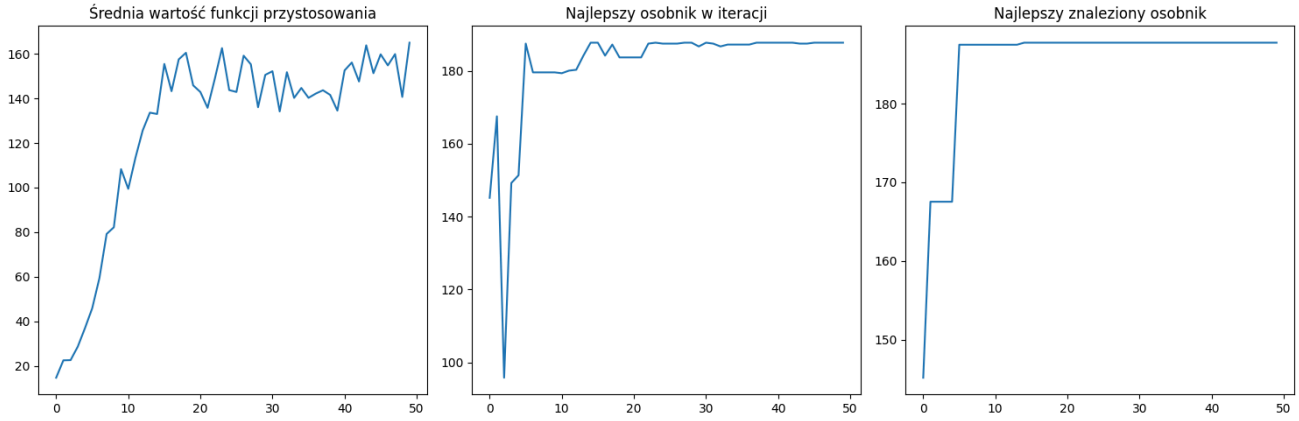
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 30
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465



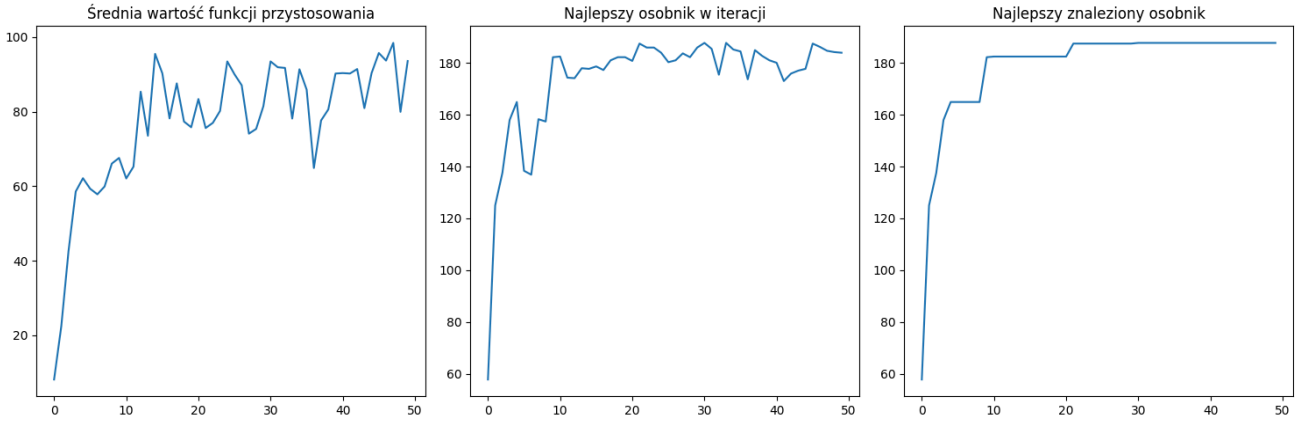
```
CODING: gray | MUTATION: 0.5 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 11
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



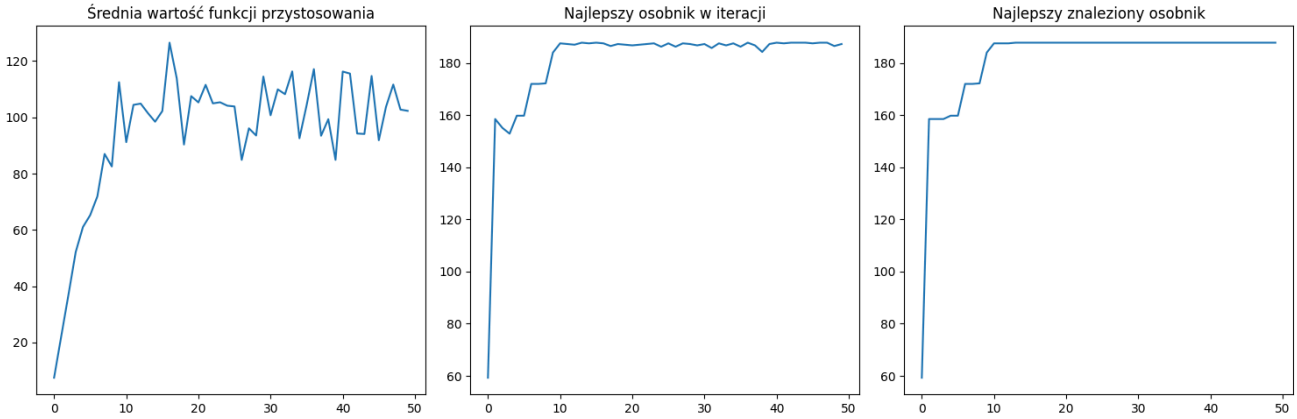

```
CODING: gray | MUTATION: 0.5 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.71499
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 14
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```



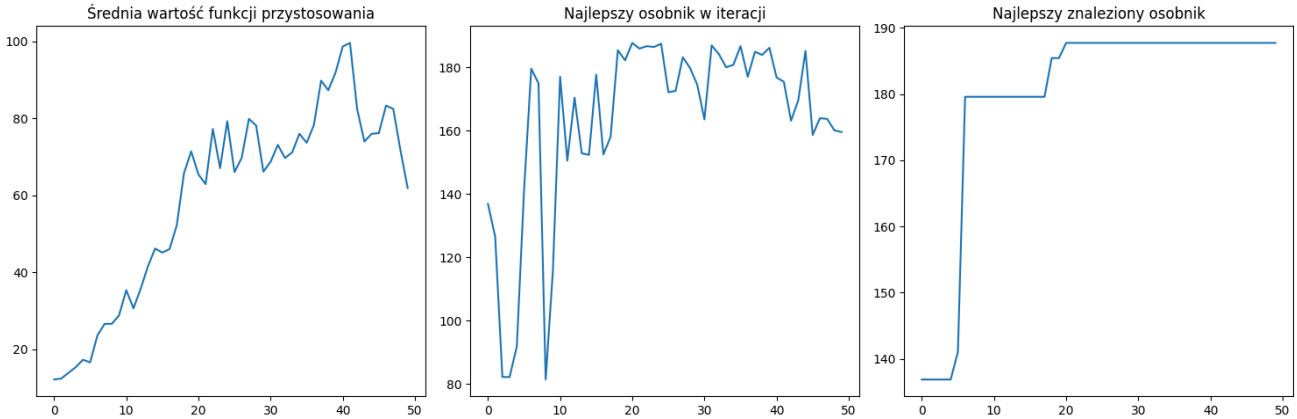

```
CODING: gray | MUTATION: 1.0 | SELECTION: roulette | GAMMA: roulette
Average best adaptation over 10 runs: 187.71499
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 30
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```




```
CODING: gray | MUTATION: 1.0 | SELECTION: threshold | GAMMA: 30
Average best adaptation over 10 runs: 187.74080
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 13
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```




```
CODING: gray | MUTATION: 1.0 | SELECTION: threshold | GAMMA: 60
Average best adaptation over 10 runs: 187.58611
---
Best solution genes: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Decoded value (x): 0.01
Found in iteration: 20
Largest function value found: 187.74079946548358
True maximum function value: 187.740799465
```

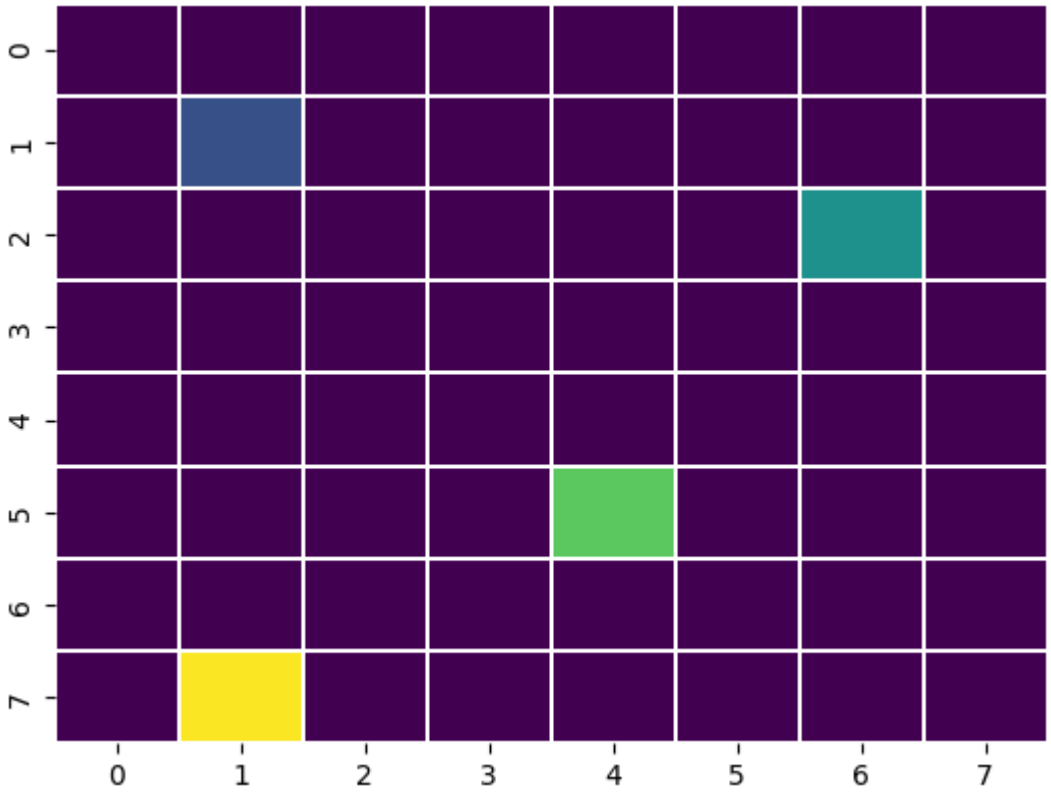


Wnioski:

- Zastosownie algorytmu genetycznego pozwoliło na znalezienie maksimum funkcji z wysoką dokładnością
- Kodowanie binarne osiągało szybciej najwyższe wartości funkcji celu, zwłaszcza przy wyższych prawdopodobieństwach mutacji
- Niskie prawdopodobieństwo mutacji prowadzi do uzyskania niskiej średniej adaptacji (np. 130.43946, 97.97408) dla początkowych iteracji (brak różnorodności prowadzi do stagnacji)
- Wyższe wartości prawdopodobieństwa zajścia mutacji dają lepsze średnie adaptacje i szybsze osiąganie maksimum
- Threshold selection lepiej sprawdza się w warunkach wysokiej zmienności (duża mutacja), natomiast roulette może być lepsza przy mniejszych wartościach mutacji
- Zwiększenie GAMMA z 30 do 60 spowalnia znalezienie wyniku
- W każdej konfiguracji, najlepsze rozwiązanie to genotyp [0, 0, ..., 0], który daje x = 0.01, co odpowiada wartości funkcji 187.740799465

Zadanie 2

W mieście, reprezentowanym za pomocą kwadratowej siatki, operujemy siecią pizzerii. Ich lokalizacje znajdują się w kwadratach o pozycjach (1,1), (2,6), (5,4), (7,1), albo jak pokazano na obrazku poniżej.



Zyski sieci zależą od budżetu wyłożonego na cztery pizzerie (B_1, B_2, B_3, B_4), gdzie każdy budżet jest w zakresie od **0 do 400** (nie może wyjść poza te wartości). Zysk jest liczony w następujący sposób:

$$Z = \sum_{i=0}^{8} \sum_{j=0}^{8} \frac{z_{i,j}^1 + z_{i,j}^2 + z_{i,j}^3 + z_{i,j}^4}{4} - (B_1 + B_2 + B_3 + B_4)^{1.15}$$

Gdzie $z_{i,j}^k$ jest dochodem k-tej pizzerii na polu (i, j). Innymi słowy, sumujemy **średnie zyski** z danego pola dla wszystkich pizzerii i odejmujemy od tego skorygowaną (potęgowaną) sumę budżetów — ze względu na konieczność rozbudowy lokali, większe trudności w utrzymaniu itp.

Zysk k-tej pizzerii na polu (i, j) liczymy natomiast następująco:

Najpierw liczymy d, czyli odległość taksówkarską (taxicab, Manhattan Distance) między pizzerią a tym polem.

- Jeżeli d jest **mniejsza** niż 2:

$$z = 1.3 \cdot \frac{B}{0.5d + 4}$$

- Jeżeli d jest **większa lub równa** 2:

$$z = \frac{B}{0.5d + 4}$$

Proszę zaprojektować **algorytm genetyczny**, który otrzyma jak najwyższą wartość funkcji $Z(B_1, B_2, B_3, B_4)$ i przedstawić, jak radzi sobie przy 10 wywołaniach (jak wygląda średnia wyniku i odchylenie standardowe).

Proszę samodzielnie wybrać **kodowanie** (rzeczywistoliczbowe, całkowitoliczbowe, binarne), wybrać **algorytm mutacji i selekcji**, oraz przeprowadzić **testy parametrów**.

Algorytm genetyczny opiera się na kodowaniu rzeczywistoliczbowym, krzyżowaniu jednopunktowym i mutacji gaussowskiej (lepsza dla wartości ciągłych). Przetestowane zostaną mechanizmy selekcji ruletkowej i progowej.

```
import numpy as np
import random

# stałe
GRID_SIZE = 9
BUDGET_MIN = 0
BUDGET_MAX = 400
NUM_PIZZERIAS = 4
PIZZERIA_POSITIONS = [(1, 1), (2, 6), (5, 4), (7, 1)]
POPULATION_SIZE = 50
NUM_GENERATIONS = 100
MUTATION_STD = 20

def profit_single_pizzeria(x, y, pizzeria_pos, B):
    d = abs(pizzeria_pos[0] - x) + abs(pizzeria_pos[1] - y)
    if d < 2:
        return 1.3 * B / (0.5 * d + 4)
    else:
        return B / (0.5 * d + 4)

def total_profit(budgets):
    total_income = 0
    for i in range(GRID_SIZE):
        for j in range(GRID_SIZE):
            field_income = sum(profit_single_pizzeria(i, j, PIZZERIA_POSITIONS[k], budgets[k]) for k in range(NUM_PIZZERIAS))
            total_income += field_income / NUM_PIZZERIAS
    total_cost = sum(budgets) ** 1.15
    return total_income - total_cost

def initialize_population():
    return [np.random.uniform(BUDGET_MIN, BUDGET_MAX, size=NUM_PIZZERIAS) for _ in range(POPULATION_SIZE)]

def roulette_selection(population, fitnesses):
```

```
    total_fit = sum(fitnesses)
    probs = [f / total_fit for f in fitnesses]
    return population[np.random.choice(len(population), p=probs)]

def threshold_selection(population, fitnesses, gamma):
    threshold = np.percentile(fitnesses, gamma * 100)
    filtered = [(ind, fit) for ind, fit in zip(population, fitnesses) if fit >= threshold]
    if not filtered:
        filtered = list(zip(population, fitnesses))
    selected = random.choice(filtered)[0]
    return selected

def one_point_crossover(p1, p2):
    point = random.randint(1, NUM_PIZZERIAS - 1)
    child = np.concatenate((p1[:point], p2[point:]))
    return np.clip(child, BUDGET_MIN, BUDGET_MAX)

def mutate(individual, mutation_rate):
    if random.random() < mutation_rate:
        idx = random.randint(0, NUM_PIZZERIAS - 1)
        individual[idx] += np.random.normal(0, MUTATION_STD)
        individual[idx] = np.clip(individual[idx], BUDGET_MIN, BUDGET_MAX)
    return individual

def run_genetic_algorithm(mutation_rate=0.2, selection_method='roulette', gamma=0.7):
    population = initialize_population()

    for _ in range(NUM_GENERATIONS):
        fitnesses = [total_profit(ind) for ind in population]
        new_population = []
        for _ in range(POPULATION_SIZE):
            if selection_method == 'roulette':  # selekcja ruletkowa
                p1 = roulette_selection(population, fitnesses)
                p2 = roulette_selection(population, fitnesses)
            else:  # selekcja progowa
                p1 = threshold_selection(population, fitnesses, gamma)
                p2 = threshold_selection(population, fitnesses, gamma)

            child = one_point_crossover(p1, p2)
            child = mutate(child, mutation_rate)
            new_population.append(child)
        population = new_population
    best = max(population, key=total_profit)
    return total_profit(best), best


# testy
configs = [
    {'mutation_rate': 0.1, 'selection_method': 'roulette'},
    {'mutation_rate': 0.2, 'selection_method': 'roulette'},
    {'mutation_rate': 0.5, 'selection_method': 'roulette'},
    {'mutation_rate': 0.8, 'selection_method': 'roulette'},
    {'mutation_rate': 1.0, 'selection_method': 'roulette'},
    {'mutation_rate': 0.1, 'selection_method': 'threshold', 'gamma': 0.3},
    {'mutation_rate': 0.2, 'selection_method': 'threshold', 'gamma': 0.3},
    {'mutation_rate': 0.5, 'selection_method': 'threshold', 'gamma': 0.3},
    {'mutation_rate': 0.8, 'selection_method': 'threshold', 'gamma': 0.3},
    {'mutation_rate': 0.1, 'selection_method': 'threshold', 'gamma': 0.7},
    {'mutation_rate': 0.2, 'selection_method': 'threshold', 'gamma': 0.7},
    {'mutation_rate': 0.5, 'selection_method': 'threshold', 'gamma': 0.7},
    {'mutation_rate': 0.8, 'selection_method': 'threshold', 'gamma': 0.7}
]

for config in configs:
    print("\nKonfiguracja:", config)
    results = []
    solutions = []
    for run in range(10):
        result, solution = run_genetic_algorithm(**config)
        results.append(result)
        solutions.append(solution)
        print(f"Run {run+1}: Zysk = {result:.4f} | B1={solution[0]:.2f}, B2={solution[1]:.2f}, B3={solution[2]:.2f}, B4={solution[3]:.
    mean = np.mean(results)
    std = np.std(results)
    best_idx = np.argmax(results)
    print(f"Średni zysk: {mean:.4f}")
    print(f"Odchylenie standardowe: {std:.4f}")
    print(f"Najlepszy wynik: {results[best_idx]:.4f} | budżety: B1={solutions[best_idx][0]:.2f}, B2={solutions[best_idx][1]:.2f}, B3={
```

```
Konfiguracja: {'mutation_rate': 0.1, 'selection_method': 'roulette'}
Run 1: Zysk = 441.2976 | B1=85.49, B2=242.17, B3=400.00, B4=113.90
Run 2: Zysk = 459.7737 | B1=5.07, B2=342.95, B3=391.65, B4=95.42
Run 3: Zysk = 444.6515 | B1=23.51, B2=273.79, B3=360.41, B4=0.00
Run 4: Zysk = 443.0270 | B1=49.67, B2=400.00, B3=296.88, B4=36.90
Run 5: Zysk = 458.6527 | B1=104.69, B2=366.19, B3=400.00, B4=32.47
Run 6: Zysk = 458.7350 | B1=106.57, B2=347.51, B3=388.42, B4=0.00
Run 7: Zysk = 433.4383 | B1=49.39, B2=392.85, B3=296.27, B4=121.84
Run 8: Zysk = 444.0162 | B1=115.51, B2=298.78, B3=359.08, B4=11.89
Run 9: Zysk = 445.9827 | B1=35.78, B2=262.90, B3=400.00, B4=139.08
Run 10: Zysk = 453.2924 | B1=0.00, B2=333.97, B3=378.21, B4=122.02
Średni zysk: 448.2867
Odchylenie standardowe: 8.4106
Najlepszy wynik: 459.7737 | budżety: B1=5.07, B2=342.95, B3=391.65, B4=95.42

Konfiguracja: {'mutation_rate': 0.2, 'selection_method': 'roulette'}
Run 1: Zysk = 406.8457 | B1=110.71, B2=245.64, B3=270.64, B4=65.39
Run 2: Zysk = 454.4885 | B1=150.70, B2=355.22, B3=400.00, B4=10.08
Run 3: Zysk = 410.5462 | B1=163.59, B2=182.21, B3=317.00, B4=0.00
Run 4: Zysk = 460.7520 | B1=64.05, B2=315.75, B3=397.86, B4=12.59
Run 5: Zysk = 461.7031 | B1=0.00, B2=305.46, B3=400.00, B4=54.10
Run 6: Zysk = 453.7088 | B1=121.47, B2=359.25, B3=398.64, B4=43.84
Run 7: Zysk = 433.6981 | B1=162.08, B2=266.51, B3=396.78, B4=99.82
Run 8: Zysk = 469.7492 | B1=57.95, B2=386.45, B3=400.00, B4=7.17
Run 9: Zysk = 425.6980 | B1=48.00, B2=206.65, B3=356.22, B4=153.02
Run 10: Zysk = 432.7519 | B1=59.16, B2=383.81, B3=318.60, B4=147.37
Średni zysk: 440.9942
Odchylenie standardowe: 21.0670
Najlepszy wynik: 469.7492 | budżety: B1=57.95, B2=386.45, B3=400.00, B4=7.17

Konfiguracja: {'mutation_rate': 0.5, 'selection_method': 'roulette'}
Run 1: Zysk = 476.7707 | B1=1.30, B2=400.00, B3=400.00, B4=11.28
Run 2: Zysk = 474.9834 | B1=11.33, B2=383.17, B3=400.00, B4=0.00
Run 3: Zysk = 455.9326 | B1=50.61, B2=317.58, B3=400.00, B4=80.66
Run 4: Zysk = 446.6318 | B1=47.62, B2=284.40, B3=361.62, B4=7.68
Run 5: Zysk = 466.5907 | B1=17.77, B2=326.80, B3=400.00, B4=7.02
Run 6: Zysk = 459.9559 | B1=15.17, B2=313.41, B3=400.00, B4=73.82
Run 7: Zysk = 473.6050 | B1=28.64, B2=400.00, B3=400.00, B4=13.76
Run 8: Zysk = 453.0473 | B1=19.58, B2=269.04, B3=400.00, B4=87.53
Run 9: Zysk = 476.9490 | B1=0.00, B2=400.00, B3=400.00, B4=10.74
Run 10: Zysk = 457.6820 | B1=30.01, B2=384.09, B3=393.77, B4=111.59
Średni zysk: 464.2148
Odchylenie standardowe: 10.4541
Najlepszy wynik: 476.9490 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=10.74

Konfiguracja: {'mutation_rate': 0.8, 'selection_method': 'roulette'}
Run 1: Zysk = 442.6597 | B1=118.04, B2=227.96, B3=390.75, B4=20.87
Run 2: Zysk = 452.5381 | B1=114.59, B2=342.62, B3=380.74, B4=25.21
Run 3: Zysk = 477.3980 | B1=4.33, B2=398.60, B3=400.00, B4=0.00
Run 4: Zysk = 443.6510 | B1=21.36, B2=320.76, B3=399.40, B4=196.63
Run 5: Zysk = 447.7256 | B1=168.59, B2=321.43, B3=383.06, B4=0.00
Run 6: Zysk = 477.0769 | B1=8.82, B2=400.00, B3=400.00, B4=0.59
Run 7: Zysk = 461.8872 | B1=122.53, B2=384.91, B3=400.00, B4=0.00
Run 8: Zysk = 466.4225 | B1=0.00, B2=386.71, B3=386.56, B4=66.66
Run 9: Zysk = 441.9447 | B1=153.16, B2=351.79, B3=344.72, B4=11.77
Run 10: Zysk = 471.2690 | B1=0.00, B2=400.00, B3=400.00, B4=61.86
Średni zysk: 458.2573
Odchylenie standardowe: 13.5501
Najlepszy wynik: 477.3980 | budżety: B1=4.33, B2=398.60, B3=400.00, B4=0.00
```

```
Konfiguracja: {'mutation_rate': 1.0, 'selection_method': 'roulette'}
Run 1: Zysk = 465.4880 | B1=24.68, B2=384.56, B3=400.00, B4=72.56
Run 2: Zysk = 466.8573 | B1=0.00, B2=328.37, B3=395.77, B4=0.07
Run 3: Zysk = 455.8610 | B1=47.74, B2=282.79, B3=400.00, B4=45.74
Run 4: Zysk = 468.8630 | B1=51.89, B2=362.21, B3=400.00, B4=0.00
Run 5: Zysk = 462.6163 | B1=78.55, B2=358.80, B3=400.00, B4=26.05
Run 6: Zysk = 452.4315 | B1=0.00, B2=325.37, B3=352.88, B4=1.57
Run 7: Zysk = 477.6698 | B1=3.09, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 441.8069 | B1=35.76, B2=351.42, B3=342.30, B4=125.53
Run 9: Zysk = 465.7572 | B1=16.76, B2=327.03, B3=400.00, B4=21.32
Run 10: Zysk = 456.3370 | B1=113.22, B2=384.52, B3=400.00, B4=44.10
Średni zysk: 461.3688
Odchylenie standardowe: 9.5289
Najlepszy wynik: 477.6698 | budżety: B1=3.09, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.1, 'selection_method': 'threshold', 'gamma': 0.
3}
Run 1: Zysk = 472.8433 | B1=48.94, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.5958 | B1=0.00, B2=400.00, B3=400.00, B4=3.90
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 472.6545 | B1=50.53, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 476.8757
Odchylenie standardowe: 2.0665
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.2, 'selection_method': 'threshold', 'gamma': 0.
3}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.5, 'selection_method': 'threshold', 'gamma': 0.
3}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
```

```
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.8, 'selection_method': 'threshold', 'gamma': 0.
3}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.1, 'selection_method': 'threshold', 'gamma': 0.
7}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.2, 'selection_method': 'threshold', 'gamma': 0.
7}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.5, 'selection_method': 'threshold', 'gamma': 0.
7}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
```

```
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00

Konfiguracja: {'mutation_rate': 0.8, 'selection_method': 'threshold', 'gamma': 0.
7}
Run 1: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 2: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 3: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 4: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 5: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 6: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 7: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 8: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 9: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Run 10: Zysk = 477.9519 | B1=0.00, B2=400.00, B3=400.00, B4=0.00
Średni zysk: 477.9519
Odchylenie standardowe: 0.0000
Najlepszy wynik: 477.9519 | budżety: B1=0.00, B2=400.00, B3=400.00, B4=0.00
```

Wnioski:

- Selekcja ruletkowa osiągnęła najlepszy wynik na poziomie zysku 477.6698 dla parametru mutacji 1.0 i budżetów: B1=3.09, B2=400.00, B3=400.00, B4=0.00
- Dla selekcji progowej algorytm dobiera skrajne watości budżetów (0 i 400), niezależnie od zmian parametrów prawdopodobieństwa mutacji i gamma
- Osiągnięty w ten sposób zysk jest najwyższy, lecz skrajność wartości i brak zmienności wyników (niewielkie lub zerowe odchylenie standardowe) nie sugerują poprawności takiego rozwiązania
- W każdym rozwiązaniu największy budżet otrzymuje restauracja nr 3 o centralnej lokalizacji (5, 4), a następnie restauracja nr 2 znajdująca się w (2, 6)