**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

Факультет «Информатика и системы управления»

Кафедра «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 14

по курсу « Методы оптимизации »

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ. СРАВНЕНИЕ МЕТОДОВ ОТБОРА И РЕКОМБИНАЦИИ**

Студент: Яровикова А. С.

Группа: ИУ9-81Б

Преподаватель: Посевин Д. П.

Москва, 2024

СОДЕРЖАНИЕ

[ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ 3](#_Toc158799284)

[ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ 3](#_Toc158799285)

[РЕЗУЛЬТАТЫ 6](#_Toc158799286)

[ВЫВОДЫ 11](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать генетический алгоритм и проверить его работоспособность на поиске минимума функции Розенброка.

**Постановка задачи:**

1. Исследовать генетический алгоритм.
2. Реализовать метод на языке программирования Python, включая функции для вычисления фитнеса, методов рекомбинации, отбора, и мутации.

# **ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ**

Импорт библиотек:

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **random**

**import** **copy**

Функция отрисовки графиков:

**def** **draw\_plot**(history, marker):

**if** marker == **2**:

**for** population **in** history:

xs = [point[**0**] **for** point **in** population]

ys = [point[**1**] **for** point **in** population]

fig, ax = plt.subplots()

ax.scatter(xs, ys, s=**0.5**)

ax.set\_xlim([-**100**, **100**])

ax.set\_ylim([-**100**, **100**])

plt.show()

**if** marker == **3**:

**for** population **in** history:

xs = [point[**0**] **for** point **in** population]

ys = [point[**1**] **for** point **in** population]

zs = [rosenbrok(point) **for** point **in** population]

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

ax.scatter(xs, ys, zs, s=**2**)

ax.set\_xlim([-**100**, **100**])

ax.set\_ylim([-**100**, **100**])

plt.show()

Генерация популяции:

**def** **generate\_population**():

population = []

r = **100.0**

**for** i **in** range(population\_size):

population.append([random.uniform(-r, r), random.uniform(-r, r)])

**return** population

Инициализация параметров генетического алгоритма:

population\_size = **100** # Размер популяции

mutation\_rate = 1**.0** # Вероятность мутации

number\_of\_generations = **1000** # Количество поколений

tournament\_size = **50** # Размер турнира для селекции

elite\_size = **20** # Размер группы лучших особей

Функция вычисления фитнес-оценки (приспособленности):

**def** **fitness**(x):

**return** **1** / (rosenbrok(x) + **1**)

**def** **rosenbrok**(x):

**return** (**1** - x[**0**])\*\***2** + **100**\*(x[**1**] - x[**0**]\*\***2**)\*\***2**

Функция отбора турниром:

**def** **tournament\_selection**(population, tournament\_size):

tournament = random.sample(population, tournament\_size)

fitness\_scores = [fitness(x) **for** x **in** tournament]

winner\_index = np.argmax(fitness\_scores)

**return** tournament[winner\_index]

Функция отбора рулеткой:

**def** **roulette\_wheel\_selection**(population):

fitness\_scores = [fitness(x) **for** x **in** population]

total\_fitness = sum(fitness\_scores)

probabilities = [score / total\_fitness **for** score **in** fitness\_scores]

cumulative\_probabilities = np.cumsum(probabilities)

random\_number = np.random.rand()

selected\_index = np.searchsorted(cumulative\_probabilities, random\_number)

**return** population[selected\_index]

Функция дискретной рекомбинации:

**def** **discrete\_crossover**(parent1, parent2):

**if** random.random() < **0.5**:

**return** [parent1[**0**], parent2[**1**]]

**else**:

**return** [parent2[**0**], parent1[**1**]]

Функция промежуточной рекомбинации:

**def** **intermediate\_crossover**(parent1, parent2):

child = []

d = random.uniform(**0.25**, **0.3**)

**for** i **in** range(len(parent1)):

alpha = random.uniform(-d, **1** + d)

child.append(parent1[i] + alpha \* (parent2[i] - parent1[i]))

**return** child

Функция мутации:

**def** **mutation**(x):

**return** [x[**0**] + random.uniform(-**5**, **5**), x[**1**] + random.uniform(-**5**, **5**)]

Генетический алгоритм:

**def** **genetic\_algorithm**(crossover\_method, is\_mut, selection\_method, population):

population = copy.deepcopy(population)

history = [population]

**for** i **in** range(number\_of\_generations):

population.sort(key=fitness, reverse=**True**)

best\_individual = population[**0**]

# сохраняем выборку особей с лучшей приспособленностью

next\_generation = copy.deepcopy(population[:elite\_size])

**while** len(next\_generation) < population\_size:

**if** selection\_method == "t":

parent1 = tournament\_selection(population, tournament\_size)

parent2 = tournament\_selection(population, tournament\_size)

**else**:

parent1 = roulette\_wheel\_selection(population)

parent2 = roulette\_wheel\_selection(population)

child = []

**if** crossover\_method == "d":

child = discrete\_crossover(parent1, parent2)

**else**:

child = intermediate\_crossover(parent1, parent2)

**if** is\_mut:

x = mutation(child)

**else**:

x = child

next\_generation.append(x)

population = copy.deepcopy(next\_generation)

**if** i % **50** == **0**:

history.append(population)

**if** i == **0** **or** (i + **1**) % **50** == **0**:

print(f'Generation {i+1}/{number\_of\_generations}, Best Fitness: {fitness(best\_individual):.6f}')

population.sort(key=fitness, reverse=**True**)

draw\_plot(history, marker=**3**)

**return** population[**0**]

population = generate\_population()

Тестирование генетического алгоритма:

res1 = genetic\_algorithm("d", **True**, "t", population)

print("Дискретная рекомбинация: ", res1)

res2 = genetic\_algorithm("i", **True**, "t", population)

print("Промежуточная рекомбинация: ", res2)

# **РЕЗУЛЬТАТЫ**

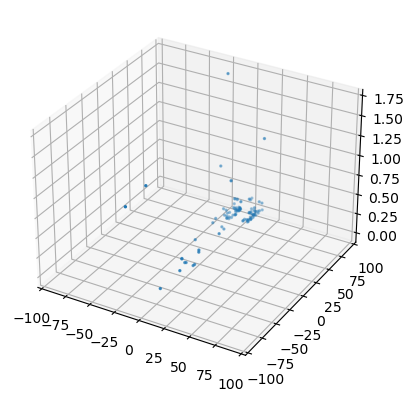


Рисунок 1 – Первое поколение генетического алгоритма для дискретной рекомбинации, отбором турниром и с учетом мутации

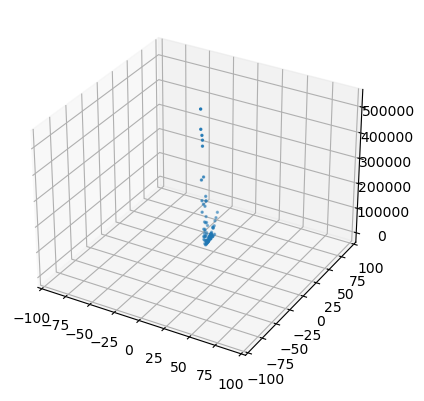


Рисунок 2 - Пятисотое поколение генетического алгоритма для дискретной рекомбинации, отбором турниром и с учетом мутации

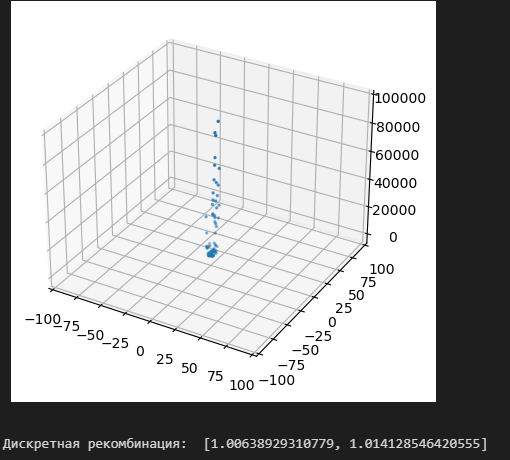


Рисунок 3 - Результат для дискретной рекомбинации, отбором турниром и с учетом мутации

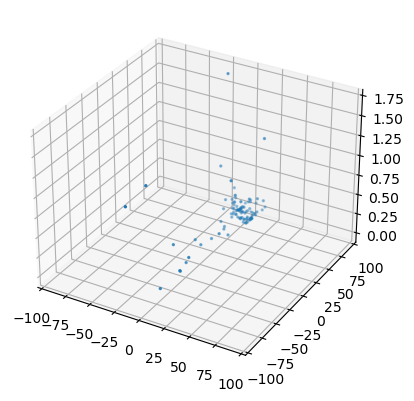


Рисунок 4 - Первое поколение генетического алгоритма для промежуточной рекомбинации, отбором турниром и с учетом мутации

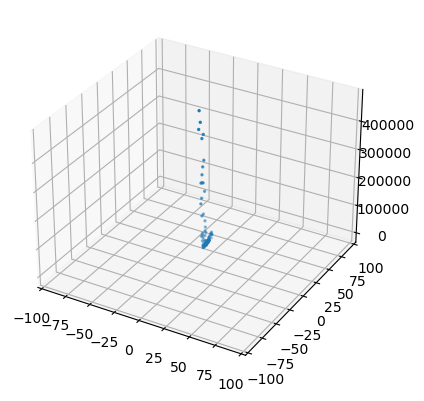


Рисунок 5 - Пятисотое поколение генетического алгоритма для промежуточной рекомбинации, отбором турниром и с учетом мутации

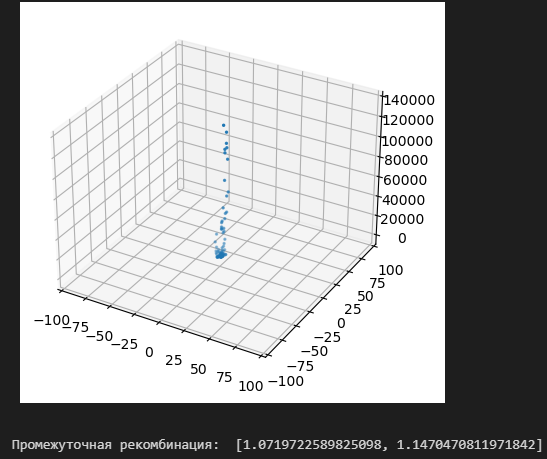


Рисунок 6 - Результат для промежуточной рекомбинации, отбором турниром и с учетом мутации

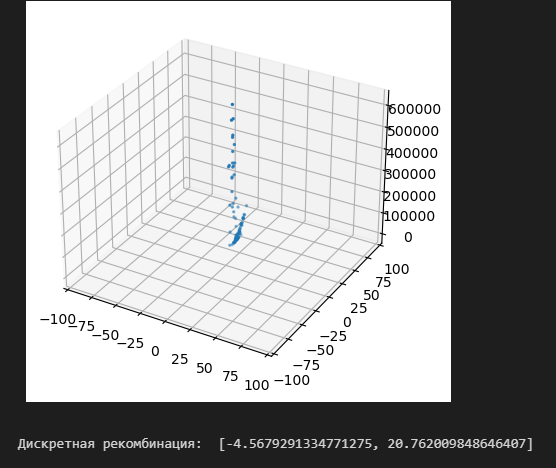


Рисунок 7 - Результат для дискретной рекомбинации, отбором рулеткой и с учетом мутации

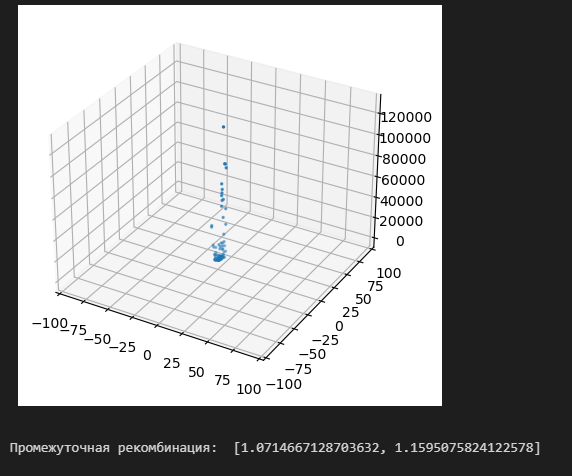


Рисунок 8 - Результат для промежуточной рекомбинации, отбором рулеткой и с учетом мутации



Рисунок 9 - Результат для дискретной рекомбинации без учета мутации



Рисунок 10 - Результат для промежуточной рекомбинации без учета мутации

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы был реализован генетический алгоритм. Работоспособность реализации проверена на функции Розенброка. Результаты тестирования показывают, что отбор рулеткой и отбор турниром дают одинаково хорошие результаты. При этом, оценивая результаты сравнения методов дискретной рекомбинации и промежуточной рекомбинации, промежуточная дает более точные результаты. Также важно отметить, что результаты без учета мутации оказались намного хуже, чем с учетом мутации, что подтверждает важность данного шага в генетическом алгоритме.