**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»

Летучка № 5

по курсу « Методы оптимизации »

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ ПОИСКА МИНИМУМА ПОЛИНОМА**

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 5](#_Toc158799286)

[ВЫВОДЫ 9](#_Toc158799287)

# **ЦЕЛЬ И ПОСТАНОВКА ЗАДАЧИ**

**Цель работы:**

Реализовать генетический алгоритм для поиска минимума полинома.

**Постановка задачи:**

1. Исследовать генетический алгоритм.
2. Реализовать метод на языке программирования Python, включая функции для генерации популяции, методов отбора, кроссовера и мутации.

# **ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ**

Импорт библиотек:

**import** **random**

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

**import** **seaborn** **as** **sns**

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

Генерация исходной популяции:

# функция генерации случайной исходной популяции в заданном диапазоне

**def** **generate\_population**(size, rrange):

r1, r2 = rrange

**return** [random.uniform(r1, r2) **for** \_ **in** range(size)]

Функция селекции:

# функция выбора пары для скрещивания

# сортировка особей из популяции на основе их приспособленности и выбор лучших особей для скрещивания в количестве num

**def** **select**(population, fitness, number\_of\_parents):

**return** sorted(population, key=fitness)[**0**:number\_of\_parents]

Функция кроссовера:

# функция скрещивания

**def** **crossover**(parents, number\_of\_children):

**return** [((random.choice(parents) + random.choice(parents)) / **2**) **for** \_ **in** range(number\_of\_children)]

Функция мутации:

# функция мутации с заданной вероятностью

**def** **mutate**(x, mutation\_rate, rrange):

r1, r2 = rrange

**if** random.random() < mutation\_rate:

**return** random.uniform(r1, r2)

**return** x

Генетический алгоритм:

**def** **genetic\_algorithm**(fitness, generations, population\_size, x\_range, mutation\_rate):

population = generate\_population(population\_size, x\_range)

best\_solution = min(population, key=fitness)

plt.figure()

x = np.linspace(\*x\_range, **100**)

plt.plot(x, fitness(x), color='black')

plt.scatter(population, np.array([fitness(p) **for** p **in** population]), color='green')

plt.show()

**for** generation **in** range(generations):

parents = select(population, fitness, population\_size // **2**)

children = crossover(parents, population\_size - len(parents))

population = parents + [mutate(child, mutation\_rate, x\_range) **for** child **in** children]

current\_best = min(population, key=fitness)

**if** fitness(current\_best) < fitness(best\_solution):

plt.figure()

x = np.linspace(\*x\_range, **100**)

plt.plot(x, fitness(x), color='black')

plt.scatter(population, np.array([fitness(p) **for** p **in** population]), color='green')

plt.scatter(best\_solution, fitness(best\_solution), color='red')

plt.show()

best\_solution = current\_best

print(f"Generation {generation}: Best solution = {best\_solution}, Score = {fitness(best\_solution)}")

**return** best\_solution

Тестирование алгоритма:

fit = **lambda** x: **5** - **24** \* x + **17** \* x\*\***2** - **11**/**3** \* x\*\***3** + **1**/**4** \* x\*\***4**

generations = **100**

population\_size = **100**

x\_range = (**0**, **11**)

mutation\_rate = **0.1**

best = genetic\_algorithm(fit, generations, population\_size, x\_range, mutation\_rate)

print(f"Best found solution: x = {best}, f(x) = {fit(best)}")

# Расширение диапозона и другая функция

fit = **lambda** x: x\*\***2** - **10**\*x + **20**

generations = **100**

population\_size = **100**

x\_range = (-**100**, **200**)

mutation\_rate = **0.1**

best = genetic\_algorithm(fit, generations, population\_size, x\_range, mutation\_rate)

print(f"Best found solution: x = {best}, f(x) = {fit(best)}")

# **РЕЗУЛЬТАТЫ**

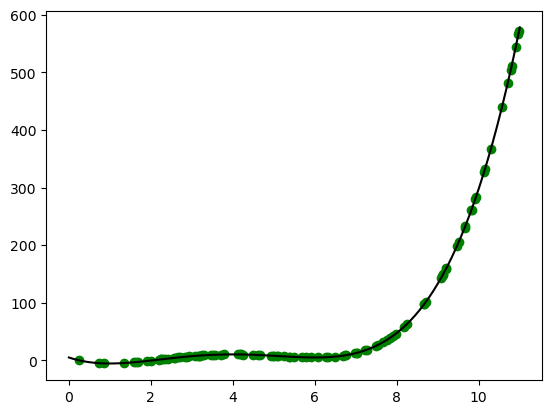


Рисунок 1 – Сгенерированная начальная случайная популяция для полинома из лекции

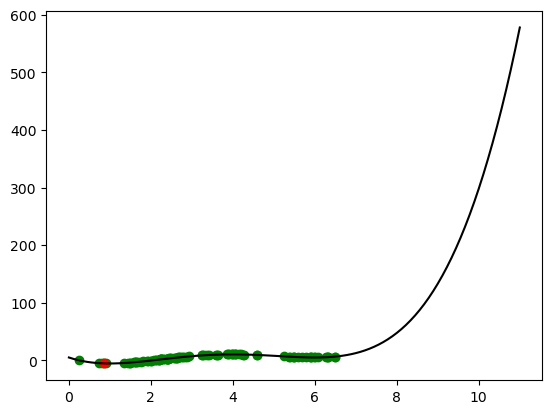


Рисунок 2 – Первое поколение генетического алгоритма для полинома из лекции

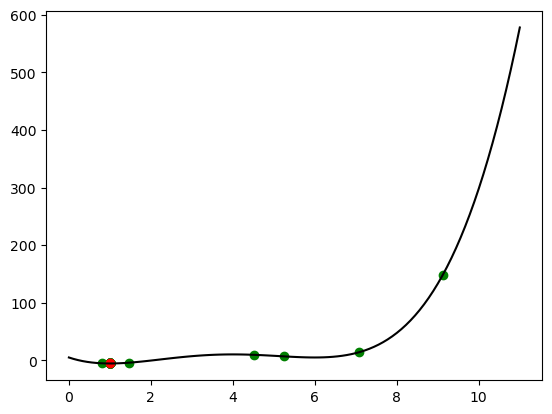


Рисунок 3 - Шестнадцатое поколение генетического алгоритма для полинома из лекции

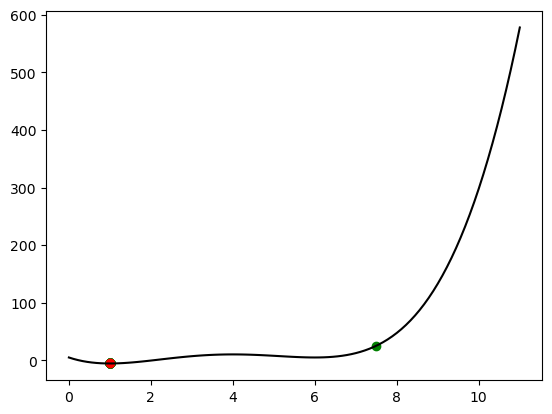


Рисунок 4 - Результат генетического алгоритма для полинома из лекции



Рисунок 5 - Результат для полинома из лекции

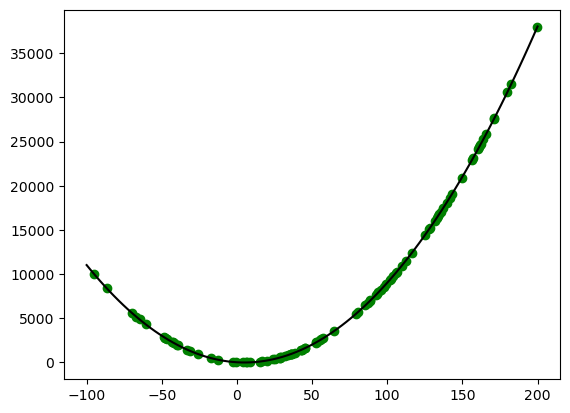


Рисунок 6 - Сгенерированная начальная случайная популяция для кастомного полинома

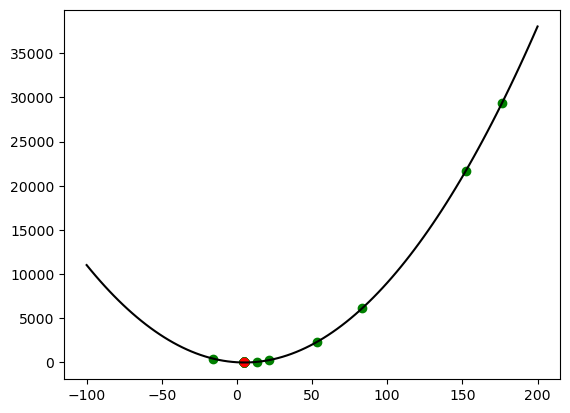


Рисунок 7 - Первое поколение генетического алгоритма для кастомного полинома

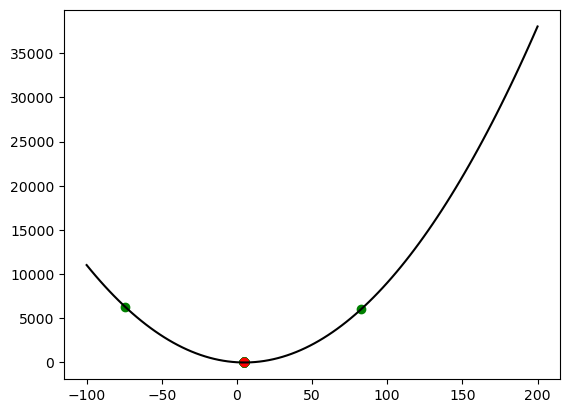


Рисунок 8 - Шестнадцатое поколение генетического алгоритма для кастомного полинома



Рисунок 9 - Результат для кастомного полинома

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы был реализован генетический алгоритм. Работоспособность реализации проверена на различных полиномах. Результаты тестирования подтверждают высокую точность алгоритма для поиска минимума полинома.