Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

**Отчет по лабораторной работе №3**

**По дисциплине**

**«Эвристические методы оптимизации»**

Студент гр. 434-М1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.А. Колпаков

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Студент гр. 434-М1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.В. Баранов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Профессор каф. АСУ, д.т.н

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Мицель

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Томск 2024

**Оглавление**

[Цель работы 2](#_Toc176630422)

[Задание на лабораторную работу 2](#_Toc176630423)

[**1 КРАТКАЯ ТЕОРИЯ** 3](#_Toc176630424)

[**2 РЕЗУЛЬТАТ РАБОТЫ** 3](#_Toc176630425)

[Вывод 5](#_Toc176630426)

# Цель работы

Научиться кодировать и декодировать целые и вещественные числа.

# Задание на лабораторную работу

Написать программу поиска минимума функции одной переменной на множестве вещественных чисел с бинарным кодированием кодом Грэя. Программа может быть написана на любом алгоритмическом языке.

Входные данные:

* целевая функция ;
* границы интервала поиска минимума функции [-7 ,7], где a и b – целые числа;
* количество особей в популяции K =10;
* количество поколений N (количество итераций) = 15;
* параметр окончания поиска ε (точность решения) = 0,001.

**1 КРАТКАЯ ТЕОРИЯ**

Ген – атомарный элемент хромосомы (может быть битом, числом или неким другим объектом.

Хромосома (цепочка) – упорядоченная последовательность генов (строка из каких-либо чисел). Если эта строка представлена бинарной строкой из нулей и единиц, например, 101010, то она получена либо с использованием двоичного кодирования, либо кода Грея. Каждая позиция хромосомы называется геном.

Генотип(код) – упорядоченная последовательность хромосом (одно из решений).

Фенотип – набор значений, соответствующих генотипу.

Особь (индивидуум) – конкретный экземпляр генотипа (вариант решения задачи).

Особи представляются хромосомами с закодированными в них множествами параметров задачи, т.е. решений. Обычно особь состоит из одной хромосомы, поэтому в дальнейшем особь и хромосома идентичные понятия.

Популяция – набор особей (набор решений задачи). В начале алгоритма случайным образом генерируется начальная популяция (набор решений). Эти решения будут становиться лучше (эволюционировать) в процессе работы алгоритма до тех пор, пока не удовлетворят условиям задачи.

Кроссинговер (кроссовер) – операция, при которой две хромосомы обмениваются своими частями. Например, 1100&1010 → 1110&1000. Здесь произошел обмен вторым геном (вторым младшим разрядом).

Мутация – случайное изменение одной или нескольких позиций в хромосоме. Например, 1010011 →1010001.

Инверсия – изменение порядка следования битов в хромосоме или в ее фрагменте. Например, 1100 → 0011.

Рекомбинация – операция, при которой две хромосомы обмениваются своими частями.

Функция приспособленности. Одним из центральных понятий является функция приспособленности или функция пригодности. Она оценивает то, насколько приспособлена данная особь в популяции, другими словами, она определяет качество особей популяции.

**2 РЕЗУЛЬТАТ РАБОТЫ**

Ниже представлен листинг программы на языке Python:

import random

import matplotlib.pyplot as plt

import numpy as np

import csv

import os

L = 16  # Длина строки

def target\_function(x):

    return x\*\*4 - 10\*(x\*\*3) + 36 \* x\*\*2 + 5 \* x

def binary\_to\_gray(binary\_string):

    binary\_value = int(binary\_string, 2)

    gray\_value = binary\_value ^ (binary\_value >> 1)

    return f'{gray\_value:0{L}b}'

def gray\_to\_binary(gray\_string):

    gray\_value = int(gray\_string, 2)

    binary\_value = gray\_value

    shift = 1

    while gray\_value >> shift:

        binary\_value ^= gray\_value >> shift

        shift += 1

    return f'{binary\_value:0{L}b}'

def decode(gray\_string, a, b):

    binary\_string = gray\_to\_binary(gray\_string)

    r = (2\*\*L - 1)

    value = int(binary\_string, 2)

    return a + (b - a) \* value / r

def fitness(gray\_string, a, b):

    x = decode(gray\_string, a, b)

    return target\_function(x)

def selection(population, a, b):

    return sorted(population, key=lambda ind: fitness(ind, a, b))[:2]

def crossover(parent1, parent2):

    point = random.randint(1, L - 1)

    child1 = parent1[:point] + parent2[point:]

    child2 = parent2[:point] + parent1[point:]

    return child1, child2

def mutate(child):

    mutation\_prob = 1 / L  # Вероятность мутации

    child\_list = list(child)

    for i in range(L):

        if random.random() < mutation\_prob:

            child\_list[i] = '1' if child\_list[i] == '0' else '0'

    return ''.join(child\_list)

def genetic\_algorithm(a, b, K, N, epsilon):

    population = [binary\_to\_gray(''.join(random.choice('01') for \_ in range(L))) for \_ in range(K)]

    initial\_population = population.copy()

    best\_values = []

    for generation in range(N):

        best\_individuals = selection(population, a, b)

        new\_population = []

        while len(new\_population) < K:

            parent1, parent2 = random.sample(best\_individuals, 2)

            child1, child2 = crossover(parent1, parent2)

            child1 = mutate(child1)

            child2 = mutate(child2)

            new\_population.extend([child1, child2])

        population = new\_population[:K]

        best\_solution = min(population, key=lambda ind: fitness(ind, a, b))

        best\_value = fitness(best\_solution, a, b)

        best\_real\_value = decode(best\_solution, a, b)

        best\_values.append(best\_value)

        if abs(best\_value) <= epsilon:

            break

    return best\_real\_value, best\_value, best\_values, initial\_population

def save\_to\_csv(a, b, K, N, epsilon, initial\_population, best\_x, best\_f):

    file\_exists = os.path.isfile('genetic\_algorithm\_results.csv')

    with open('genetic\_algorithm\_results.csv', mode='a', newline='') as file:

        writer = csv.writer(file)

        if not file\_exists:

            writer.writerow(['a', 'b', 'K', 'N', 'epsilon', 'Best x', 'Best f(x)', 'Initial Population'])

        writer.writerow([a, b, K, N, epsilon, best\_x, best\_f, initial\_population])

def plot\_function(a, b):

    x = np.linspace(a, b, 400)

    y = target\_function(x)

    plt.plot(x, y, label="f(x)")

    plt.title("Целевая функция")

    plt.xlabel("x")

    plt.ylabel("f(x)")

    plt.grid(True)

    plt.legend()

    plt.show()

a = -7

b = 7

K = 20

N = 15

epsilon = 0.001

best\_x, best\_f, best\_values, initial\_population = genetic\_algorithm(a, b, K, N, epsilon)

save\_to\_csv(a, b, K, N, epsilon, initial\_population, best\_x, best\_f)

print(f"Лучшее решение: x = {best\_x}, f(x) = {best\_f}")

plt.scatter(best\_x, best\_f, color='red', label=f"Точка минимума({best\_x}, {best\_f})")

plot\_function(a, b)

В данной реализации добавлен график функции для наглядности.

Описание функций:

target\_function(x) – возвращает целевую функцию:

decode(binary\_string, a, b) – декодирует бинарную строку в соответствии с заданным интервалом по формуле 1).

binary\_to\_gray(binary\_string) – Преобразует двоичное число в код Грея.

def gray\_to\_binary(gray\_string) – переводит число из кода Грея в двоичный код.

fitness(binary\_string, a, b) – подставляет полученное значение декодированной бинарной строки в целевую функцию.

selection(population, a, b) – выполняет сортировку особей из популяции по наименьшему значению целевой функции.

crossover(parent1, parent2) – выполняет функцию кроссинговера между 2 родителями для создания новой популяции, для этого выбирается случайная позиция в хромосоме и делит ее на 2 части левую и правую. Так как родителя 2 мы получаем 2 поделенные хромосомы из которых формируются гены 2 потомков таким образом что :

* 1 Потомок = [левая часть хромосомы 1 родителя: правая часть хромосомы 2 родителя].
* 2 Потомок = [левая часть хромосомы 2 родителя: правая часть хромосомы 1 родителя].

mutate(child) – выполняет функцию мутации хромосомы ребенка таким образом что с вероятностью 1/L случайный ген ребенка заменится на противоположный.

genetic\_algorithm(a, b, K, N, epsilon) – непосредственно сам генетический алгоритм:

Инициализация популяции: Сначала создается начальная популяция из двоичных строк (представляющих потенциальные решения). Размер популяции — K, а каждый элемент представляет собой строку случайных битов.

Цикл поколений: Функция выполняется для определенного количества поколений, N, или до достижения заданного порога точности, epsilon.

Отбор: В каждом поколении происходит отбор лучших особей на основе функции приспособленности (fitness), которая оценивает, насколько близко решение (индивид) к оптимальному значению. Эта функция использует параметры a и b.

Кроссинговер: Из выбранных лучших особей случайным образом выбираются пары родителей. Для каждой пары выполняется операция кроссовера, при которой части генов родителей комбинируются, чтобы создать потомков.

Мутация: Потомки подвергаются мутации, где случайные биты изменяются с некоторой вероятностью, чтобы обеспечить разнообразие в популяции.

Обновление популяции: Новое поколение потомков заменяет старую популяцию, сохраняя фиксированный размер популяции K.

Лучшее решение: Для каждого поколения находится особь с наилучшей функцией приспособленности. Ее значение сохраняется, и также декодируется в реальное значение.

Проверка критерия остановки: Алгоритм останавливается, если наилучшая особь в популяции достигает порога точности, определенного параметром epsilon, или если выполнено заданное количество поколений.

Результаты: Функция возвращает лучшее найденное решение, его оценку приспособленности и историю значений приспособленности на каждом этапе.

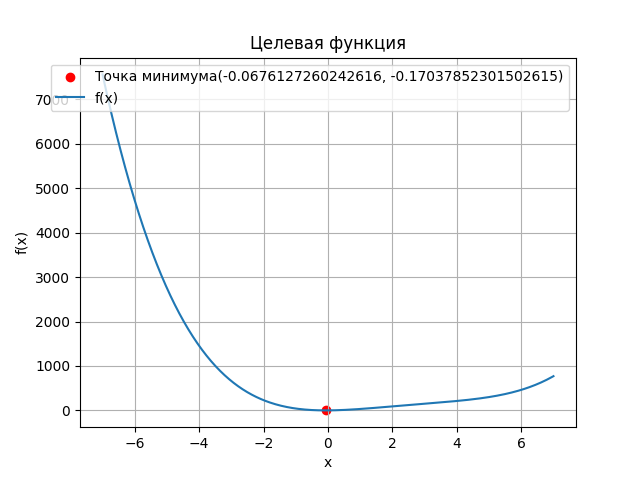


Рисунок 2.1 – Результат работы программы

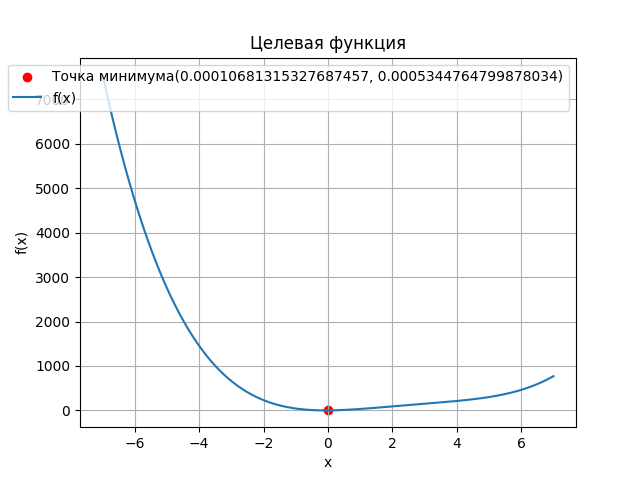


Рисунок 2.2 – Результат работы программы с бинарным кодированием

На одних и тех же входных данных кодирование Грея показывает более точный результат из-за того, что в коде Грея при переходе от одного числа к следующему изменяется только один бит, тогда как в стандартном двоичном представлении могут изменяться несколько битов одновременно. Это уменьшает вероятность больших изменений в результате при небольших мутациях, что делает пространство решений более "плавным" и менее подверженным скачкам значений.

# Вывод

Во время выполнения лабораторной работы была реализована программа для нахождения локального минимума функции, использующая генетический алгоритм, и построен график целевой функции с обозначенной точкой локального минимума. Алгоритм с использованием кода Грея оказался точнее, и демонстрирует более точное значение локального минимума функции на меньшем количестве популяций и особей в ней за счет реализации более мелкого шага.

Использованы такие генетические операторы как:

* Скрещивание.
* Мутация.
* Селекция.