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 №5**

**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 многих переменных на множестве вещественных чисел с вещественным кодированием. Программа может быть написана на любом алгоритмическом языке.

Входные данные:

* целевая функция
* границы интервала поиска минимума функции и ;
* количество особей в популяции K =10;
* количество поколений N (количество итераций) = 15;
* параметры алгоритма , β = 0,5;
* 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 numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def target\_function(x):

    x1, x2 = x

    return 2 \* x1\*\*2 + x1 \* x2 + x2\*\*2 - 3 \* x1

def generate\_population(size, dimensions, bounds):

    return [[random.uniform(bounds[i][0], bounds[i][1]) for i in range(dimensions)] for \_ in range(size)]

def crossover(parent1, parent2):

    alpha = random.uniform(0, 1)

    return [(alpha \* x1 + (1 - alpha) \* x2) for x1, x2 in zip(parent1, parent2)]

def mutate(individual, beta, mutation\_rate, bounds):

    for i in range(len(individual)):

        if random.random() < mutation\_rate:

            individual[i] += random.uniform(-beta, beta)

            individual[i] = max(min(individual[i], bounds[i][1]), bounds[i][0])

    return individual

def genetic\_algorithm(target\_function, bounds, population\_size, generations, beta, epsilon):

    dimensions = len(bounds)

    population = generate\_population(population\_size, dimensions, bounds)

    history = []

    for generation in range(generations):

        population.sort(key=target\_function)

        best\_individual = population[0]

        history.append(population.copy())

        if target\_function(best\_individual) <= epsilon:

            break

        new\_population = population[:population\_size // 2]

        while len(new\_population) < population\_size:

            parents = random.sample(population[:population\_size // 2], 2)

            child = crossover(parents[0], parents[1])

            child = mutate(child, beta, mutation\_rate=0.1, bounds=bounds)

            new\_population.append(child)

        population = new\_population

    return best\_individual, target\_function(best\_individual), history

# Построение графика

def plot\_history\_3d(history, bounds, target\_function, best\_solution):

    x1 = np.linspace(bounds[0][0], bounds[0][1], 100)

    x2 = np.linspace(bounds[1][0], bounds[1][1], 100)

    x\_opt, y\_opt = 0.857, -0.429

    z\_opt = target\_function([ x\_opt, y\_opt])

    X1, X2 = np.meshgrid(x1, x2)

    Z = target\_function([X1, X2])

    fig = plt.figure(figsize=(12, 8))

    ax = fig.add\_subplot(111, projection='3d')

    ax.plot\_surface(X1, X2, Z, cmap='viridis', alpha=0.8)

    ax.set\_xlabel('$x\_1$')

    ax.set\_ylabel('$x\_2$')

    ax.set\_zlabel('$f(x\_1, x\_2)$')

    ax.set\_title('Эволюция популяции в 3D')

    for i, population in enumerate(history):

        x\_coords = [ind[0] for ind in population]

        y\_coords = [ind[1] for ind in population]

        z\_coords = [target\_function(ind) for ind in population]

        ax.scatter(x\_coords, y\_coords, z\_coords, color='red', s=20, label=f'Поколение {i + 1}' if i == 0 else None)

        for j, (x, y, z) in enumerate(zip(x\_coords, y\_coords, z\_coords)):

            ax.text(x, y, z, f'{i + 1}:{j + 1}', fontsize=8, color='black', ha='center', va='center')

    best\_x1, best\_x2 = best\_solution

    best\_z = target\_function(best\_solution)

    ax.scatter(best\_x1, best\_x2, best\_z, color='blue', s=50, label='Лучшее решение', marker='o')

    ax.scatter(x\_opt, y\_opt, z\_opt, color='green', s=50, label='Целевое решение', marker='o')

    ax.text(best\_x1, best\_x2, best\_z, f'Best', fontsize=10, color='blue', ha='center', va='center', bbox=dict(facecolor='white', edgecolor='blue', pad=1))

    plt.legend(loc='upper right')

    plt.show()

bounds = [(-2, 2), (-2, 2)]  # Границы для x1 и x2

population\_size = 20  # Количество особей

generations = 1000  # Количество поколений

beta = 0.5  # Параметр мутации

epsilon = 0.01  # Точность решения

best\_solution, best\_value, history = genetic\_algorithm(target\_function, bounds, population\_size, generations, beta, epsilon)

print("Лучшее решение:", best\_solution)

print("Значение целевой функции:", best\_value)

plot\_history\_3d(history, bounds, target\_function, best\_solution)

В данной реализации добавлен график функции для наглядности.

Описание функций:

target\_function(x) – возвращает целевую функцию:

generate\_population(size, dimensions, bounds) - функция генерирует начальную популяцию случайных индивидов (хромосом). Каждый индивид — это список значений с длиной dimensions, в котором значения случайно выбираются из заданных границ bounds.

crossover(parent1, parent2) – выполняет функцию кроссинговера между 2 родителями для создания новой популяции. Вес каждого родителя в новом ребенке определяется случайным числом (α) от 0 до 1, которое меняется при каждом запуске.

Если α ближе к 1, первый ребенок больше похож на первого родителя, а второй — на второго.

Если α ближе к 0, наоборот: первый ребенок больше похож на второго родителя, а второй — на первого.

Когда α находится где-то посередине, оба ребенка представляют собой равномерное сочетание свойств обоих родителей.

mutate(child) – выполняет функцию мутации хромосомы ребенка таким образом что вносит случайные изменения в значение "ребенка" с заданной вероятностью. Если мутация происходит, к значению добавляется небольшая случайная величина из диапазона [−β,β]. После этого значение ограничивается в пределах [0, 1]. Это добавляет разнообразие и помогает избежать зацикливания на одном решении.

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.

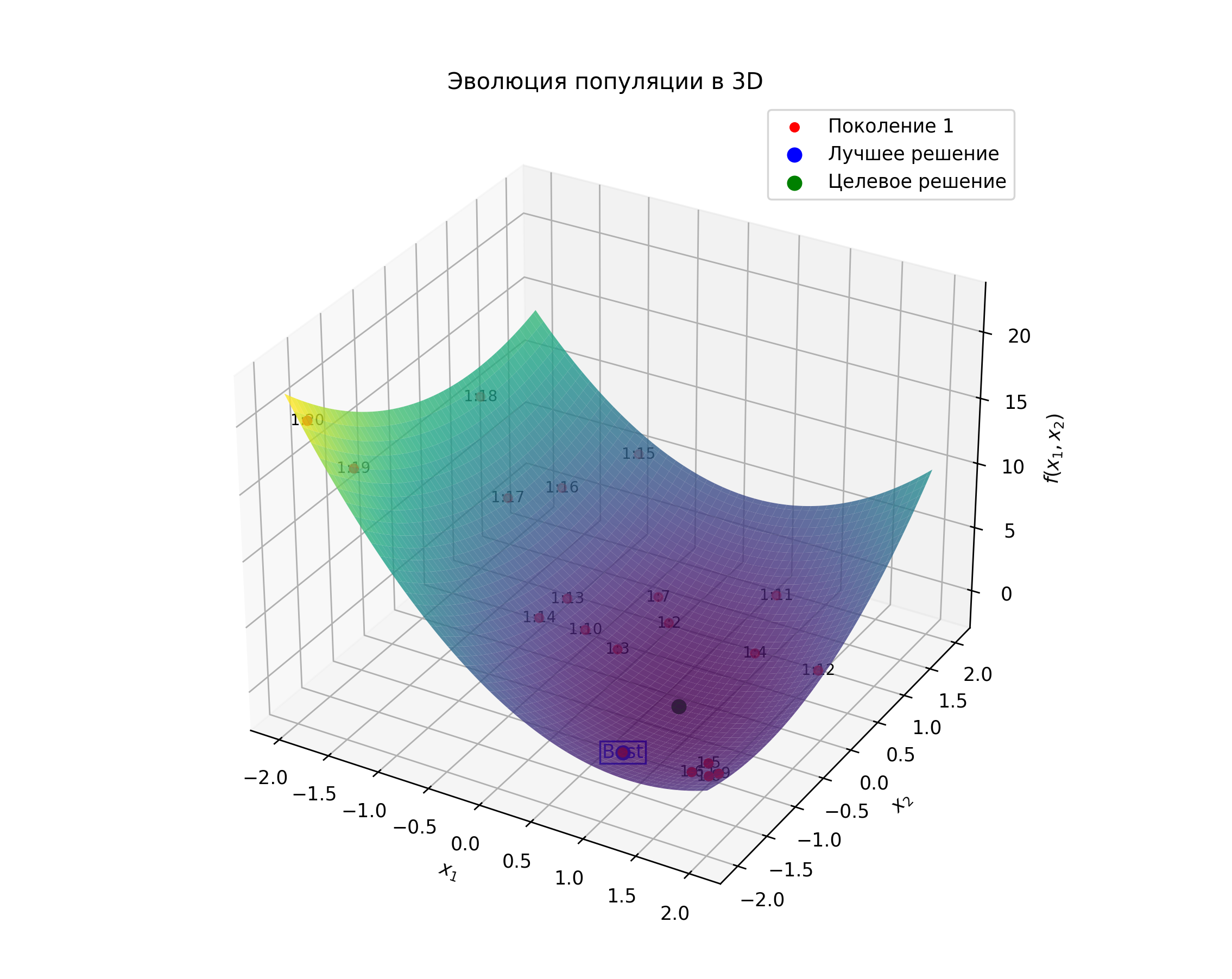


Рисунок 2.1 – Результат работы программы

# Вывод

В ходе выполнения генетического алгоритма для нахождения локального минимума двумерной функции было получено оптимальное решение, минимизирующее целевую функцию. Алгоритм применяет методы кроссинговера и мутации для создания нового поколения особей, что позволяет искать более оптимальные решения на протяжении нескольких поколений.

Результат выполнения программы продемонстрировал улучшение решения с каждым поколением, пока не было достигнуто заданное значение точности (epsilon), что свидетельствует о завершении поиска оптимального решения.

На графике показана эволюция популяции на протяжении всех поколений, где красные точки отображают особей на каждом этапе, а синие точки указывают на лучшее найденное решение. Зелёная точка показывает целевое решение, к которому стремится алгоритм. График демонстрирует, как популяция постепенно сходится к оптимальному решению, подтверждая эффективность использования генетического алгоритма для решения задачи оптимизации.

Использованы такие генетические операторы как:

* Скрещивание.
* Мутация.
* Селекция.