МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра вычислительных технологий**

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

**по дисциплине Методы поисковой оптимизации**

Работу выполнила\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ромашкина А.А.

Факультет Компьютерных технологий и прикладной математики

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии курс 4

Краснодар

2022

СОДЕРЖАНИЕ

[Задание 3](#_Toc121773993)

[Алгоритм 3](#_Toc121773994)

[Результат работы программы 3](#_Toc121773995)

[Листинг 5](#_Toc121773996)

Тема работы: Алгоритм иммунной сети.

# Задание

Необходимо разработать алгоритм иммунной сети для оптимизации функции Розенброкка.

*.*

Она имеет глобальный минимум в точке , где .

# Алгоритм

1. Генерация начальных популяций антител и антигенов.
2. На каждой итерации популяция антител изменяется:
   1. Для каждого антигена:
      1. Вычисляется популяция лучших по аффинности антител.
      2. Выполняется клонирование и мутация антител популяции:
         1. Для каждого антитела создается популяция клонов и их мутация.
         2. Клоны сортируются по аффинности. Берутся только первые nd особи.
         3. Из полученной популяции удаляются все клоны, аффинность которых меньше bb.
         4. Популяция урезается по минимальному значению аффинности равной br.
      3. Полученная популяция дописывается к текущей популяции антител.
      4. Текущая популяция антител сортируется по аффинности и уменьшается; выкидываются все антитела, аффинность которых меньше br.
3. Из конечной популяции выбирается особь с минимальным значением целевой функции.

# Результат работы программы

Вся работа программы представлена на рисунках 1-2 ниже.

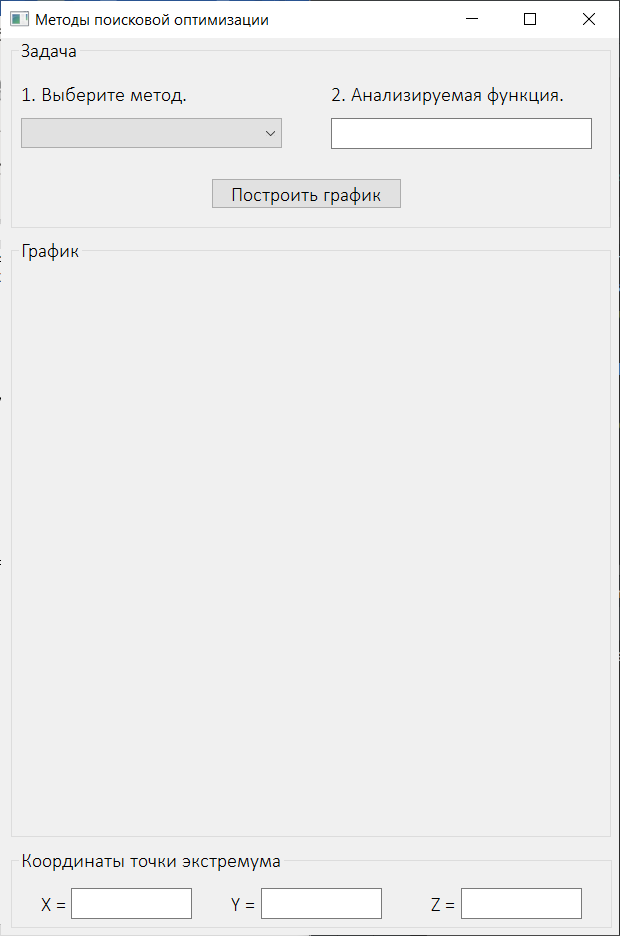


Рисунок 1 – Начальный вид программы при запуске

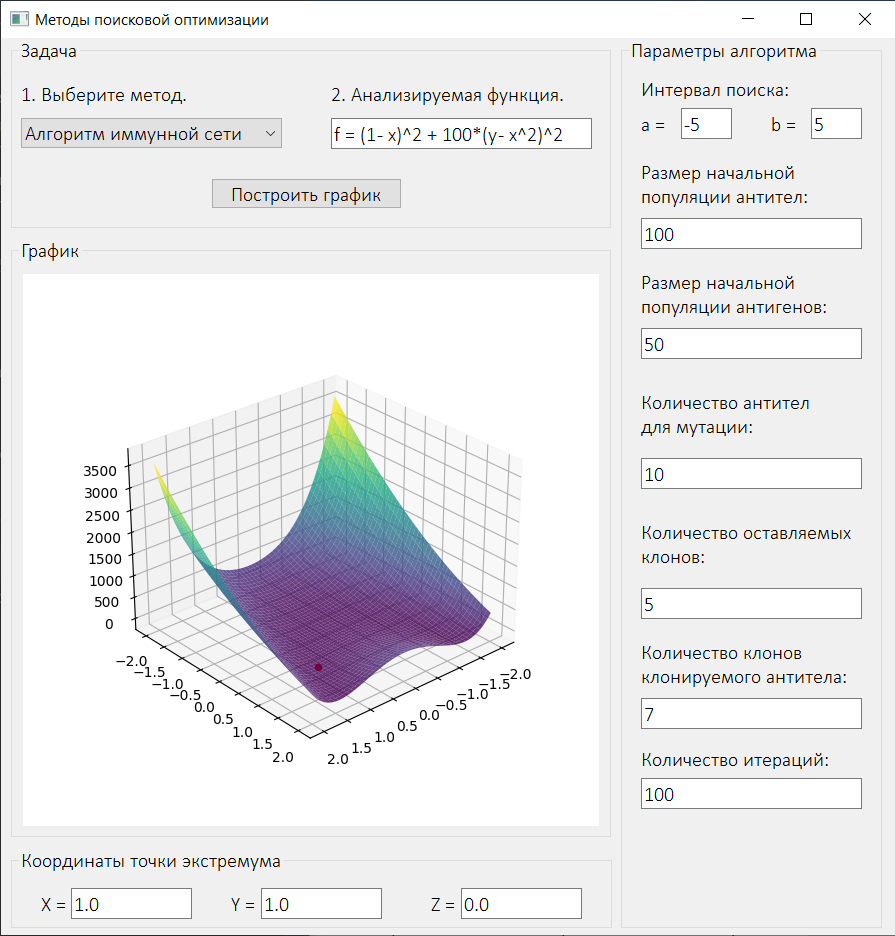


Рисунок 2 – Вид программы при выборе метода градиентного спуска с постоянным шагом

# Листинг

import math

import random

# Класс антитела (параметры - координаты точки (x, y))

class Antibody:

def \_\_init\_\_(self, x\_value, y\_value):

self.x = x\_value

self.y = y\_value

# Класс антигена (параметры - координаты точки (x, y))

class Antigen:

def \_\_init\_\_(self, x\_value, y\_value):

self.x = x\_value

self.y = y\_value

# Класс популяции (параметры - список особей (антител или антигенов)

# a - минимальное значение из области поиска

# b - максимальное значение из области поиска

# population\_size - размер популяции

# min\_affinity - минимальное значение аффинности

# f - функция (см.функцию Rosenbrok в конце этого файла)

class Population:

def \_\_init\_\_(self):

self.individuals = None

# Создание популяции антител

def create\_population\_of\_antibodies(self, population\_size, a, b):

self.individuals = [Antibody(random.uniform(a, b), random.uniform(a, b)) for \_ in range(population\_size)]

# Создание популяции антигенов

def create\_population\_of\_antigens(self, population\_size, a, b):

self.individuals = [Antigen(random.uniform(a, b), random.uniform(a, b)) for \_ in range(population\_size)]

# Метод, который вырезает из популяции особей со значением аффинности

# меньшей, чем min\_affinity

def reduce\_population(self, min\_affinity, f):

flag = True

population\_size = len(self.individuals)

while flag:

flag = False

for i in range(population\_size):

for j in range(i + 1, population\_size):

if self.individuals[i] is not None and self.individuals[j] is not None:

if affinity(self.individuals[i], self.individuals[j]) < min\_affinity:

flag = True

if f([self.individuals[i].x, self.individuals[i].y]) < f([self.individuals[j].x, self.individuals[j].y]):

self.individuals[j] = None

else:

self.individuals[i] = None

self.individuals = list(filter(lambda a: a is not None, self.individuals))

# Вспомогаетльный метод: переводит популяцию в список особей

def to\_list(self):

list\_individuals\_of\_population = []

for i in self.individuals:

list\_individuals\_of\_population.append([i.x, i.y])

return list\_individuals\_of\_population

# Основной класс. Реализация алгоритма иммунной сети.

# f - функция (см.функцию Rosenbrok в конце этого файла)

# population\_of\_antibodies - популяция антител

# population\_of\_antigens - популяция антигенов

# nb - количество антител, которые будут подвергнуты мутации

# nc - число клонов клонируемого антитела

# nd - число оставляемых клонов

# bb - пороговый коэффициент гибели

# br - коэффициент клонального сжатия

# iterations - количество итераций алгоритма

class ImmuneNetworkAlgorithm:

def \_\_init\_\_(self, f):

self.f = f

self.population\_of\_antibodies = None

self.population\_of\_antigens = None

# Метод, вычисляющий популяцию клеток памяти

def create\_memory\_cells(self, nb, nc, nd, bb, br):

for antigen in self.population\_of\_antigens.individuals:

best\_antibodies = self.get\_best\_antibodies(antigen, nb)

population\_memory\_cells = self.clone\_and\_mutate(best\_antibodies, nc, nd, antigen, bb, br)

for memory\_cells in population\_memory\_cells.individuals:

self.population\_of\_antibodies.individuals.append(memory\_cells)

self.population\_of\_antibodies.reduce\_population(br, self.f)

# Метод, который отбирает из популяции антител лучших по аффинности

def get\_best\_antibodies(self, antigen, nb):

population\_of\_best\_antibodies = Population()

population\_of\_best\_antibodies.individuals = []

self.population\_of\_antibodies.individuals.sort(key=lambda x: affinity(x, antigen), reverse=False)

for i in self.population\_of\_antibodies.individuals[:nb]:

population\_of\_best\_antibodies.individuals.append(i)

return population\_of\_best\_antibodies

# Метод, который выполняет клонирование и мутацию антител

def clone\_and\_mutate(self, population\_of\_antibodies, nc, nd, antigen, bb, br):

clone\_population = Population()

clone\_population.individuals = []

for antibody in population\_of\_antibodies.individuals:

clone\_population.individuals = []

# Коэффициент мутации

alpha = math.exp(-0.1 \* affinity(antibody, antigen))

for c in range(nc):

clone\_population.individuals.append(Antibody(antibody.x + alpha \* random.uniform(-0.5, 0.5), antibody.y + alpha \* random.uniform(-0.5, 0.5)))

clone\_population.individuals.sort(key=lambda a: affinity(a, antigen), reverse=False)

population\_memory\_cells = Population()

population\_memory\_cells.individuals = clone\_population.individuals[:nd]

for i in range(nd):

if affinity(population\_memory\_cells.individuals[i], antigen) < bb:

population\_memory\_cells.individuals = population\_memory\_cells.individuals[:i]

break

population\_memory\_cells.reduce\_population(br, self.f)

return population\_memory\_cells

# Основной метод алгоритма иммунной сети

def immune\_network\_algorithm(self, a, b, size\_population\_of\_antibodies, size\_population\_of\_antigens, nb, nd, nc, iterations, bb, br):

self.population\_of\_antibodies = Population()

self.population\_of\_antibodies.create\_population\_of\_antibodies(size\_population\_of\_antibodies, a, b)

self.population\_of\_antigens = Population()

self.population\_of\_antigens.create\_population\_of\_antigens(size\_population\_of\_antigens, a, b)

# print(self.population\_of\_antibodies.to\_list())

for i in range(iterations):

self.create\_memory\_cells(nb, nc, nd, bb, br)

# print(self.population\_of\_antibodies.to\_list())

mn = self.f([self.population\_of\_antibodies.individuals[0].x, self.population\_of\_antibodies.individuals[0].y])

idx = 0

for i in range(1, len(self.population\_of\_antibodies.individuals)):

if self.f([self.population\_of\_antibodies.individuals[i].x, self.population\_of\_antibodies.individuals[i].y]) < mn:

idx = i

mn = self.f([self.population\_of\_antibodies.individuals[i].x, self.population\_of\_antibodies.individuals[i].y])

return [self.population\_of\_antibodies.individuals[idx].x, self.population\_of\_antibodies.individuals[idx].y]

# Метод, который вычисляет значение аффинности, т.е. евклидово расстояние между двумя точками

def affinity(individual1, individual2):

return (individual1.x - individual2.x)\*\*2 + (individual1.y - individual2.y)\*\*2