МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по лабораторной работе №4 по курсу**

**«МЕТОДЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ»**

Работу выполнили

Студенты 36 группы

Агаджанян А.С.

Гаранина Л.В.

Преподаватель:

Нигодин Е.А.

Краснодар 2025

**Цель работы:** разработать алгоритм роя частиц оптимизации функции Растригина.

**Ход работы:**

Алгоритм роя частиц был предложен в 1995 году Джеймсом Кеннеди (Kennedy) и Расселом Еберхартом (Eberhart). Идея алгоритма была частично заимствована из исследований поведения скоплений животных (косяков рыб, стай птиц и т. п.), модель была немного упрощена и добавлены элементы поведения толпы людей, поэтому, в отличие, например, от алгоритма пчел агенты алгоритма (возможные решения) были названы нейтрально - частицы.

Блок-схема алгоритма представлена на рисунке 1.

Изображение выглядит как текст, чек, снимок экрана, диаграмма

Автоматически созданное описание

Рисунок 1 – Блок-схема алгоритма роя частиц.

Алгоритм роя частиц работает в -мерном пространстве (область поиска), в котором рыщут частицы (агенты алгоритма). В начале частицы разбросаны случайным образом по всей области поиска и каждая частица имеет случайный вектор скорости. В каждой точке, где побывала частица, рассчитывается значение целевой функции. При этом каждая частица запоминает, какое лучшее значение целевой функции она лично нашла и где, а также каждая частица знает расположение точки, являющейся лучшей среди всех точек, которые разведали частицы. На каждой итерации частицы корректируют свою скорость (модуль и направление), чтобы быть поближе к лучшей точке, которую частица нашла сама, и в то же время, чтобы приблизиться к точке, которая в данный момент является глобально лучшей. Через некоторое количество итераций частицы должны собраться вблизи лучшей точки, хотя, вполне вероятно, что часть частиц останется где-то в относительно неплохом локальном экстремуме, но необходимо, чтобы хотя бы одна частица оказалась вблизи глобального экстремума.

Одна из особенностей алгоритма — это коррекция скорости. Именно от этого шага зависит сходимость алгоритма. В первоначальном виде алгоритма коррекция скорости выглядела следующим образом:

где – i-я компонента скорости при t-ой итерации алгоритма – i-я координата частицы при t-ой итерации алгоритма – i-я координата лучшего решения, найденного частицей – i-я координата лучшего решения, найденного всеми частицами , – случайные числа в интервале (0, 1) , – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Затем корректируется текущая координата каждой частицы:

После этого рассчитывается значение целевой функции в каждой новой точке. Каждая частица проверяет, не стала ли новая координата лучшей среди всех точек, где она побывала. Затем среди всех новых точек проверяется, не найдена ли новая глобально лучшая точка. В этом случае запоминаются её координаты и значение целевой функции в ней.

Одна из модификаций алгоритма состоит в том, чтобы добавить еще один весовой коэффициент перед текущей скоростью (коэффициент инерции), благодаря которому скорость изменяется более плавно:

Этот коэффициент может быть константой, а может зависеть от номера итерации , например, линейно уменьшаться, начиная от небольшой величины, меньшей 1, и до какой-то другой величины, отличной от нуля.

К недостаткам алгоритма относится, что он обладает всеми недостатками, что и другие стохастические алгоритмы оптимизации – отсутствует гарантия сходимости алгоритма при конечном числе частиц, отсюда и увеличение необходимого количества расчетов целевой функции (но, очевидно, меньше, чем у простого перебора). Недостаток характеризующий именно этот алгоритм – это ориентация на одну лучшую точку, что увеличивает вероятность остановки алгоритма в неплохом, но не на глобальном минимуме.

К достоинствам алгоритма относится его понятность и возможность быстро перестроить алгоритм на другую формулу для расчета скорости частиц.

**Особенности реализации генетического алгоритма.**

Для создания программы использовался язык программирования Python. Для графической визуализации были подключены графические фреймворки Tkinter и Matplotlib.

Интерфейс программы имеет вид:

Изображение выглядит как диаграмма, текст, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «4», ввести размер роя (количество частиц), количество итераций, альфа, бета, инерция и величину задержки.

В поле «Выполнение и результаты» динамически выводятся результаты работы алгоритма в виде шагов, представленных координатами и значениями оптимизируемой функции в этих координатах. На рисунке 2 показано поле «Выполнение и результаты» до запуска программы.

Изображение выглядит как текст, снимок экрана, Прямоугольник, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Выполнение и результаты.

На рисунке 3 показана панель «Функция и отображение её графика»

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Функция и отображение её графика.

Для запуска алгоритма необходимо нажать кнопку «Выполнить», при этом поле «Выполнение и результаты» начнёт динамически заполняться точками, а на отображаемой функции можно наглядно увидеть функционирование алгоритма - искомые точки, которые также отображаются динамически как показано на рисунке 4. Результирующая точка выделена на графике крестом.

Изображение выглядит как текст, снимок экрана, диаграмма, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Результат работы программы.

**Вывод:** в ходе работы были изучен алгоритм роя частиц, а затем разработан алгоритм оптимизации функции Растригина.

**Листинг программы**

Файл swarm.py

import tkinter as tk

from tkinter import ttk

from tkinter import scrolledtext

import numpy as np

import math

import numpy

from abc import ABCMeta, abstractmethod

import numpy

import numpy.random

class Particle (object):

    """

    Класс, описывающий одну частицу

    """

    def \_\_init\_\_ (self, swarm):

        """

        swarm - экземпляр класса Swarm, хранящий параметры алгоритма, список частиц и лучшее значение роя в целом

        position - начальное положение частицы (список)

        """

        # Текущее положение частицы

        self.\_\_currentPosition = self.\_\_getInitPosition (swarm)

        # Лучшее положение частицы

        self.\_\_localBestPosition = self.\_\_currentPosition[:]

        # Лучшее значение целевой функции

        self.\_\_localBestFinalFunc = swarm.getFinalFunc (self.\_\_currentPosition)

        self.\_\_velocity = self.\_\_getInitVelocity (swarm)

    @property

    def position (self):

        return self.\_\_currentPosition

    @property

    def velocity (self):

        return self.\_\_velocity

    def \_\_getInitPosition (self, swarm):

        """

        Возвращает список со случайными координатами для заданного интервала изменений

        """

        return numpy.random.rand (swarm.dimension) \* (swarm.maxvalues - swarm.minvalues) + swarm.minvalues

    def \_\_getInitVelocity (self, swarm):

        """

        Сгенерировать начальную случайную скорость

        """

        assert len (swarm.minvalues) == len (self.\_\_currentPosition)

        assert len (swarm.maxvalues) == len (self.\_\_currentPosition)

        minval = -(swarm.maxvalues - swarm.minvalues)

        maxval = (swarm.maxvalues - swarm.minvalues)

        return numpy.random.rand (swarm.dimension) \* (maxval - minval) + minval

    def nextIteration (self, swarm):

        # Случайный вектор для коррекции скорости с учетом лучшей позиции данной частицы

        rnd\_currentBestPosition = numpy.random.rand (swarm.dimension)

        # Случайный вектор для коррекции скорости с учетом лучшей глобальной позиции всех частиц

        rnd\_globalBestPosition = numpy.random.rand (swarm.dimension)

        veloRatio = swarm.localVelocityRatio + swarm.globalVelocityRatio

        commonRatio = (2.0 \* swarm.currentVelocityRatio /

                (numpy.abs (2.0 - veloRatio - numpy.sqrt (veloRatio \*\* 2 - 4.0 \* veloRatio) ) ) )

        # Посчитать новую скорость

        newVelocity\_part1 = commonRatio \* self.\_\_velocity

        newVelocity\_part2 = (commonRatio \*

                swarm.localVelocityRatio \*

                rnd\_currentBestPosition \*

                (self.\_\_localBestPosition - self.\_\_currentPosition) )

        newVelocity\_part3 = (commonRatio \*

                swarm.globalVelocityRatio \*

                rnd\_globalBestPosition \*

                (swarm.globalBestPosition - self.\_\_currentPosition) )

        self.\_\_velocity = newVelocity\_part1 + newVelocity\_part2 + newVelocity\_part3

        # Обновить позицию частицы

        self.\_\_currentPosition += self.\_\_velocity

        finalFunc = swarm.getFinalFunc (self.\_\_currentPosition)

class Swarm (object):

    """

    Базовый класс для роя частиц. Его надо переопределять для конкретной целевой функции

    """

    \_\_metaclass\_\_ = ABCMeta

    def \_\_init\_\_ (self,

            swarmsize,

            minvalues,

            maxvalues,

            currentVelocityRatio,

            localVelocityRatio,

            globalVelocityRatio):

        """

        swarmsize - размер роя (количество частиц)

        minvalues - список, задающий минимальные значения для каждой координаты частицы

        maxvalues - список, задающий максимальные значения для каждой координаты частицы

        currentVelocityRatio - общий масштабирующий коэффициент для скорости

        localVelocityRatio - коэффициент, задающий влияние лучшей точки, найденной частицей на будущую скорость

        globalVelocityRatio - коэффициент, задающий влияние лучшей точки, найденной всеми частицами на будущую скорость

        """

        self.\_\_swarmsize = swarmsize

        assert len (minvalues) == len (maxvalues)

        assert (localVelocityRatio + globalVelocityRatio) > 4

        self.\_\_minvalues = numpy.array (minvalues[:])

        self.\_\_maxvalues = numpy.array (maxvalues[:])

        self.\_\_currentVelocityRatio = currentVelocityRatio

        self.\_\_localVelocityRatio = localVelocityRatio

        self.\_\_globalVelocityRatio = globalVelocityRatio

        self.\_\_globalBestFinalFunc = None

        self.\_\_globalBestPosition = None

        self.\_\_swarm = self.\_\_createSwarm ()

    def \_\_getitem\_\_ (self, index):

        """

        Возвращает частицу с заданным номером

        """

        return self.\_\_swarm[index]

    def \_\_createSwarm (self):

        """

        Создать рой из частиц со случайными координатами

        """

        return [Particle (self) for \_ in range (self.\_\_swarmsize) ]

    def nextIteration (self):

        """

        Выполнить следующую итерацию алгоритма

        """

        for particle in self.\_\_swarm:

            particle.nextIteration (self)

    @property

    def minvalues (self):

        return self.\_\_minvalues

    @property

    def maxvalues (self):

        return self.\_\_maxvalues

    @property

    def currentVelocityRatio (self):

        return self.\_\_currentVelocityRatio

    @property

    def localVelocityRatio (self):

        return self.\_\_localVelocityRatio

    @property

    def globalVelocityRatio (self):

        return self.\_\_globalVelocityRatio

    @property

    def globalBestPosition (self):

        return self.\_\_globalBestPosition

    @property

    def globalBestFinalFunc (self):

        return self.\_\_globalBestFinalFunc

    def getFinalFunc (self, position):

        assert len (position) == len (self.minvalues)

        finalFunc = self.\_finalFunc (position)

        if (self.\_\_globalBestFinalFunc == None or

                finalFunc < self.\_\_globalBestFinalFunc):

            self.\_\_globalBestFinalFunc = finalFunc

            self.\_\_globalBestPosition = position[:]

    @abstractmethod

    def \_finalFunc (self, position):

        pass

    @property

    def dimension (self):

        """

        Возвращает текущую размерность задачи

        """

        return len (self.minvalues)

    def \_getPenalty (self, position, ratio):

        """

        Рассчитать штрафную функцию

        position - координаты, для которых рассчитывается штраф

        ratio - вес штрафа

        """

        penalty1 = sum ([ratio \* abs (coord - minval)

            for coord, minval in zip (position, self.minvalues)

            if coord < minval ] )

        penalty2 = sum ([ratio \* abs (coord - maxval)

            for coord, maxval in zip (position, self.maxvalues)

            if coord > maxval ] )

        return penalty1 + penalty2

class Swarm\_Rastrigin (Swarm):

    def \_\_init\_\_ (self,

            swarmsize,

            minvalues,

            maxvalues,

            currentVelocityRatio,

            localVelocityRatio,

            globalVelocityRatio):

       Swarm.\_\_init\_\_ (self,

            swarmsize,

            minvalues,

            maxvalues,

            currentVelocityRatio,

            localVelocityRatio,

            globalVelocityRatio)

    def \_finalFunc (self, position):

        function = 10.0 \* len (self.minvalues) + sum (position \* position - 10.0 \* numpy.cos (2 \* numpy.pi \* position) )

        penalty = self.\_getPenalty (position, 10000.0)

        return function + penalty

# инерция

# альфа

# бетта

def printResult (swarm, iteration):

    template = u""" Лучшие координаты: {bestpos}\n Лучший результат: {finalfunc}\n\n"""

    result = template.format (iter = iteration,

            bestpos = swarm.globalBestPosition,

            finalfunc = swarm.globalBestFinalFunc)

    return result

def ParticleSwarmAlgorithm(frame,root,ax,canvas):

        def rastrigin(\*X):

            A = 10

            return A + sum([(x \*\* 2 - A \* np.cos(2 \* math.pi \* x)) for x in X])

        def run\_optimization():

            # Генерация сетки для графика целевой функции

            X = np.linspace(-4, 4, 200)

            Y = np.linspace(-4, 4, 200)

            X, Y = np.meshgrid(X, Y)

            Z = rastrigin(X, Y)

            iterCount = iteration.get()

            dimension = 3

            swarmsize = particle.get()

            minvalues = numpy.array ([-5.12] \* dimension)

            maxvalues = numpy.array ([5.12] \* dimension)

            currentVelocityRatio = inertia.get()

            localVelocityRatio = alpha.get()

            globalVelocityRatio = beta.get()

            swarm = Swarm\_Rastrigin(swarmsize,

                                    minvalues,

                                    maxvalues,

                                    currentVelocityRatio,

                                    localVelocityRatio,

                                    globalVelocityRatio

                                    )

            #для записи результатов

            results = []

            results\_text.config(state=tk.NORMAL)

            results\_text.delete(1.0, tk.END)

            for n in range(iterCount):

                ax.cla()

                # Построение поверхности графика целевой функции

                ax.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.5)

                ax.set\_xlabel('X')

                ax.set\_ylabel('Y')

                ax.set\_zlabel('Z')

                ax.set\_xticks(np.arange(-4, 4 + 1, 2))

                ax.set\_yticks(np.arange(-4, 4 + 1, 2))

                ax.set\_title("Алгоритм Роя Частиц")

                ax.scatter(swarm[0].position[0], swarm[0].position[1], swarm[0].position[2], color='red',

                               s=10)

                results\_text.insert(tk.END,

                                    f"Итерация {n}\n")

                results\_text.insert(tk.END,

                                    f"Позиция {swarm[0].position}\n")

                results\_text.insert(tk.END,

                                    f"Скорость {swarm[0].velocity}\n")

                results\_text.insert(tk.END,printResult(swarm, n))

                swarm.nextIteration()

                results\_text.yview\_moveto(1)

                canvas.draw()

                root.update()

        param\_frame2 = frame

        bg\_color = "#3d6466"

        param\_frame2.configure(bg=bg\_color)

        # Параметры задачи

        ttk.Label(param\_frame2, text="Инициализация значений", font=("Helvetica", 12,"bold")).grid(row=0, column=0, pady=15)

        ttk.Label(param\_frame2, text="Частиц", font=("Helvetica", 10)).grid(row=2, column=0)

        ttk.Label(param\_frame2, text="Итераций", font=("Helvetica", 10)).grid(row=1, column=0)

        ttk.Label(param\_frame2, text="Альфа", font=("Helvetica", 10)).grid(row=3, column=0)

        ttk.Label(param\_frame2, text="Бета", font=("Helvetica", 10)).grid(row=4, column=0)

        ttk.Label(param\_frame2, text="Инерция", font=("Helvetica", 10)).grid(row=5, column=0)

        #частиц

        particle = tk.IntVar(value=2000)

        iteration = tk.IntVar(value=100)

        alpha=tk.IntVar(value=2)

        beta=tk.IntVar(value=5)

        inertia=tk.DoubleVar(value=0.5)

        particle\_entry = ttk.Entry(param\_frame2, textvariable=particle)

        iteration\_entry = ttk.Entry(param\_frame2, textvariable=iteration)

        alpha\_entry = ttk.Entry(param\_frame2, textvariable=alpha)

        beta\_entry = ttk.Entry(param\_frame2, textvariable=beta)

        inertia\_entry = ttk.Entry(param\_frame2, textvariable=inertia)

        particle\_entry.grid(row=2, column=1)

        iteration\_entry.grid(row=1, column=1)

        alpha\_entry.grid(row=3, column=1)

        beta\_entry.grid(row=4, column=1)

        inertia\_entry.grid(row=5, column=1)

        separator = ttk.Separator(param\_frame2, orient="horizontal")  # Горизонтальная полоса разделения

        separator.grid(row=7, column=0, columnspan=2, sticky="ew", pady=10)

        # Параметры функции

        ttk.Label(param\_frame2, text="Функция и отображение ее графика", font=("Helvetica", 12, "bold")).grid(row=9, column=0, pady=10)

        ttk.Label(param\_frame2, text="Выберите функцию", font=("Helvetica", 10)).grid(row=10, column=0)

        function\_choices = ["Функция Растригина"]

        function\_var = tk.StringVar(value=function\_choices[0])

        function\_menu = ttk.Combobox(param\_frame2, textvariable=function\_var, values=function\_choices, width=22)

        function\_menu.grid(row=10, column=1, pady=5)

        separator = ttk.Separator(param\_frame2, orient="horizontal")  # Горизонтальная полоса разделения

        separator.grid(row=18, column=0, columnspan=2, sticky="ew", pady=10)

        # Создание кнопки Выполнить

        button\_style = ttk.Style()

        button\_style.configure("My.TButton", font=("Helvetica", 14))

        # Создание кнопки Выполнить

        apply\_settings\_button = ttk.Button(param\_frame2, text="Выполнить",command=run\_optimization, style="My.TButton")

        apply\_settings\_button.grid(row=21, column=1, padx=10, pady=10)

        ttk.Label(param\_frame2, text="Выполнение и результаты", font=("Helvetica", 12, "bold")).grid(row=20, column=0, pady=10)

        results\_text = scrolledtext.ScrolledText(param\_frame2, wrap=tk.WORD, height=18, width=40, padx=2, state=tk.DISABLED)

        results\_text.grid(row=21, column=0, padx=10)