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 46 группы

Стасюк К.В.

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2023

**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.

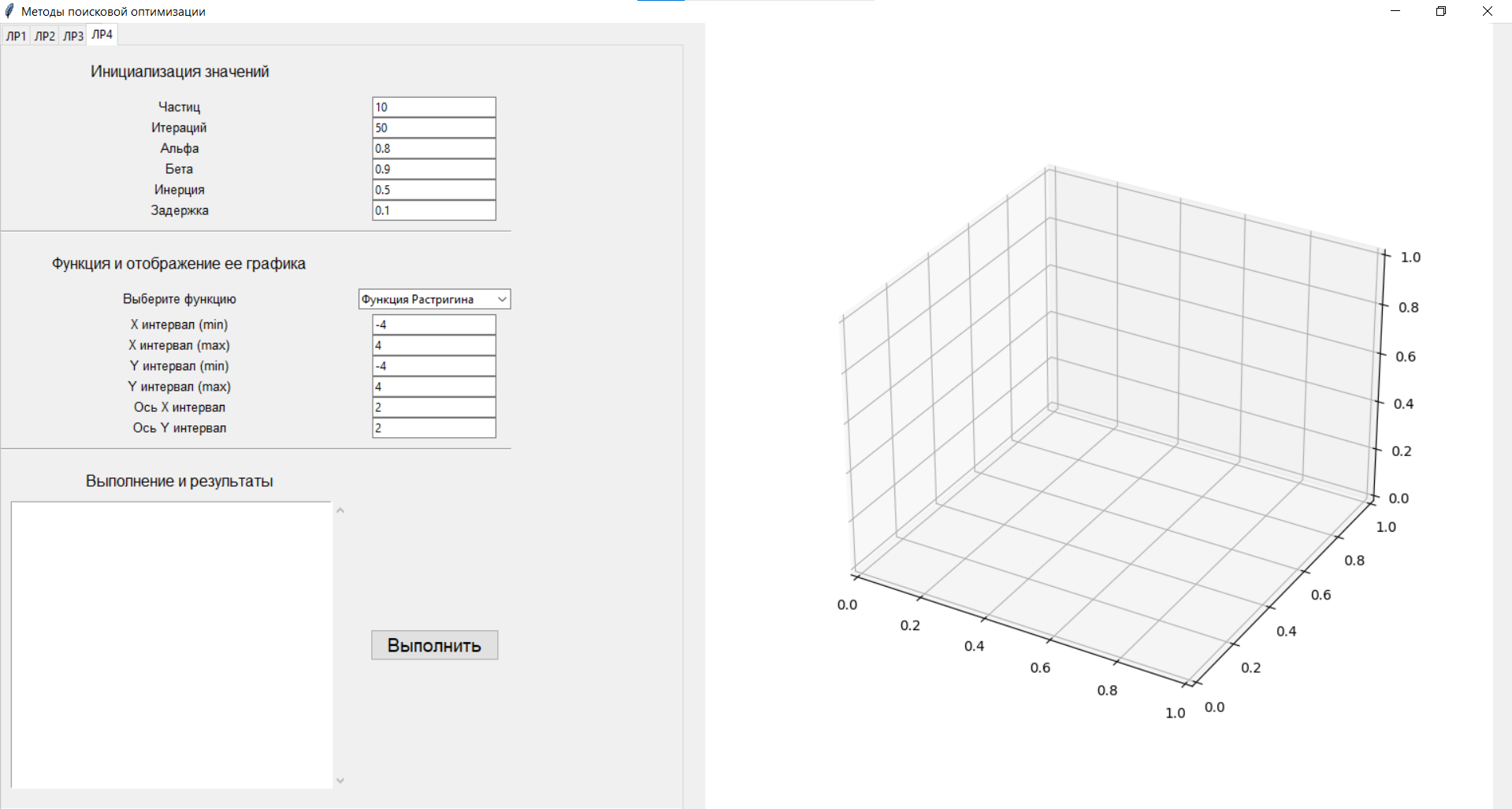
К недостаткам алгоритма относится, что он обладает всеми недостатками, что и другие стохастические алгоритмы оптимизации – отсутствует гарантия сходимости алгоритма при конечном числе частиц, отсюда и увеличение необходимого количества расчетов целевой функции (но, очевидно, меньше, чем у простого перебора). 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 3.10 и среда разработки PyCharm Professional 2023.2. Для графической визуализации были подключены графические фреймворки Tkinter и Matplotlib.

Интерфейс программы имеет вид:

Рисунок 1 – Главное окно программы.

В данном окне можно выбрать необходимую вкладку соответствующей лабораторной работы, в данном случае «ЛР4», 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

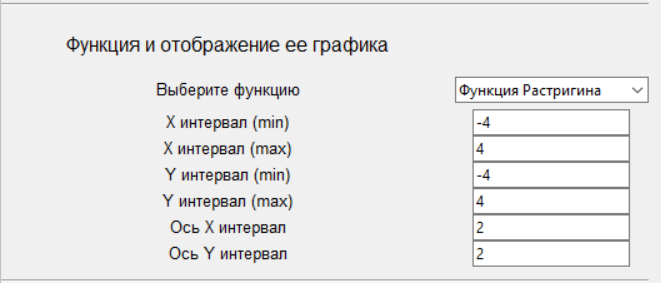
Автоматически созданное описание

Рисунок 2 – Выполнение и результаты.

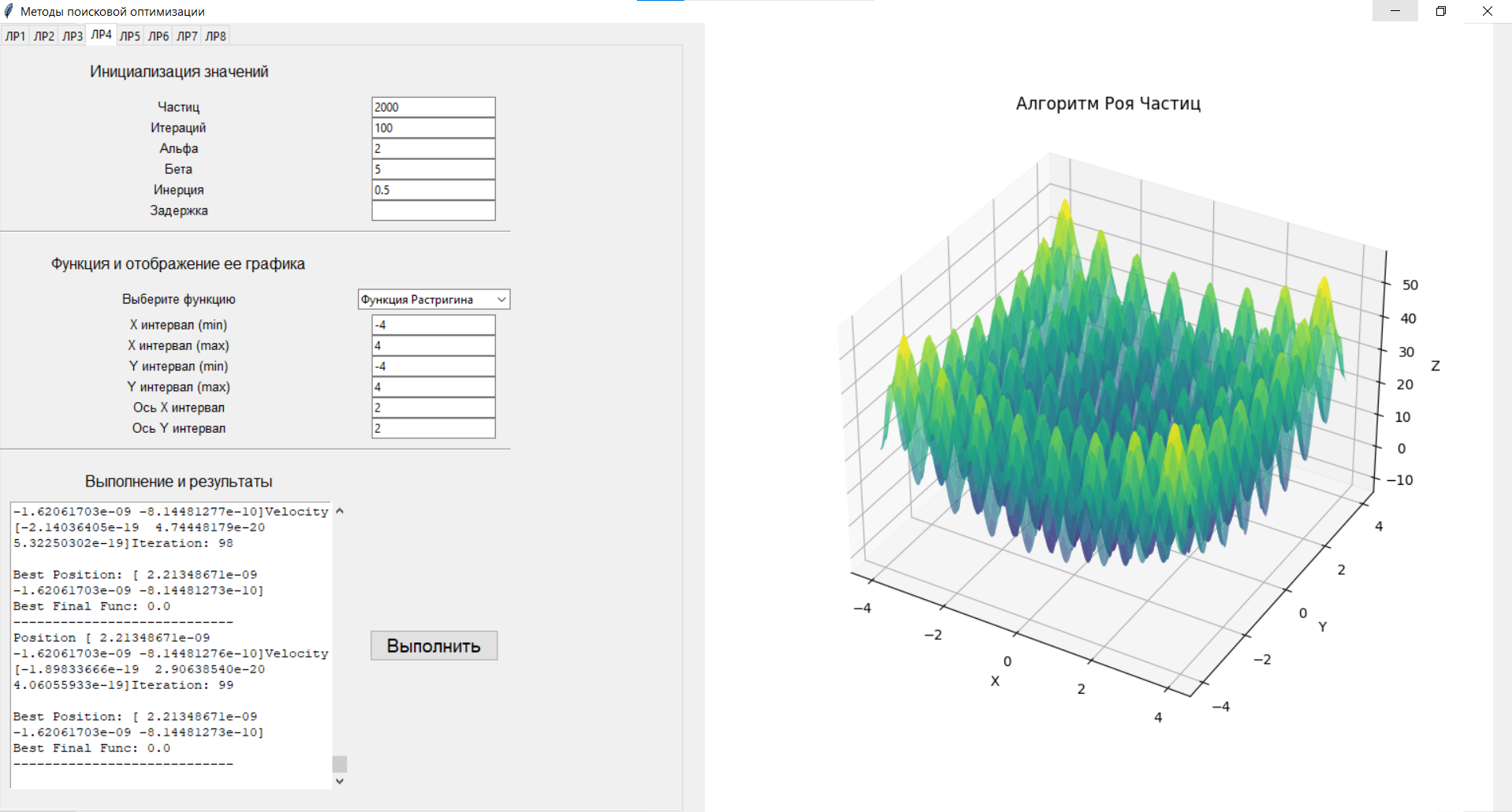
Также, в интерфейсе программы есть панель управления «Функция и отображение её графика», в котором можно выбрать необходимую функцию и задать её параметры:

1. X интервал (интервал на котором строится и минимизируется функция);
2. Y интервал (интервал на котором строится и минимизируется функция);
3. Ось X интервал (размер отображаемой сетки по оси X);
4. Ось Y интервал (размер отображаемой сетки по оси Y).

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

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. Значения результирующей функции имеют разный окрас в зависимости от положения: самые высокие точки показаны ярко-жёлтым цветом, а самые низкие – тёмно-синим.

Рисунок 4 – Результат работы программы.

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

**Листинг программы**

Файл LR4.py

import tkinter as tk

from tkinter import ttk

from tkinter import scrolledtext

import numpy as np

import math

from LR4.particleswarm.swarm import Swarm

import numpy

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"""Iteration: {iter}

Best Position: {bestpos}

Best Final Func: {finalfunc}

----------------------------

"""

result = template.format (iter = iteration,

bestpos = swarm.globalBestPosition,

finalfunc = swarm.globalBestFinalFunc)

return result

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

# Функция Розенброка для оптимизации

def rastrigin(\*X):

A = 10

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

# # Оператор селекции (выбор лучших особей)

# def selection(population, fitness\_scores):

# # Выбираем двух наилучших особей

# best\_indices = np.argsort(fitness\_scores)[:2]

# return [population[i] for i in best\_indices]

#

# # Оператор кроссовера (одноточечный кроссовер)

# def crossover(parent1, parent2):

# crossover\_point = np.random.randint(1, len(parent1))

# child = np.hstack((parent1[:crossover\_point], parent2[crossover\_point:]))

# return child

#

# # Оператор мутации

# def mutate(individual, mutation\_rate):

# mutation\_indices = np.random.rand(len(individual)) < mutation\_rate

# individual[mutation\_indices] += np.random.uniform(-0.5, 0.5)

# return individual

def run\_optimization():

# Генерация сетки для графика целевой функции

X = np.linspace(x\_interval\_min.get(), x\_interval\_max.get(), 200)

Y = np.linspace(y\_interval\_min.get(), y\_interval\_max.get(), 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=0.7)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

ax.set\_xticks(np.arange(x\_interval\_min.get(), x\_interval\_max.get() + 1, x\_axis\_interval.get()))

ax.set\_yticks(np.arange(y\_interval\_min.get(), y\_interval\_max.get() + 1, y\_axis\_interval.get()))

ax.set\_title("Алгоритм Роя Частиц")

# for i in range(len(fitness\_scores)):

# best\_individual = population[i]

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

s=10)

# print("Position", swarm[0].position)

# print("Velocity", swarm[0].velocity)

# print(printResult(swarm, n))

results\_text.insert(tk.END,

f"Position {swarm[0].position}")

results\_text.insert(tk.END,

f"Velocity {swarm[0].velocity}")

# ax.scatter(best\_solution[0], best\_solution[1], best\_fitness\_value, color='black', marker='x', s=60)

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

swarm.nextIteration()

results\_text.yview\_moveto(1)

canvas.draw()

root.update()

# # Нахождение лучшего решения после всех итераций

# final\_fitness\_scores = np.array([rastrigin(x, y) for x, y in population])

# best\_index = np.argmin(final\_fitness\_scores)

# best\_solution = population[best\_index]

# best\_fitness\_value = final\_fitness\_scores[best\_index]

#

# results\_text.insert(tk.END,

# f"\nОптимизация завершена. Лучшее решение - {best\_solution}, Значение функции - {best\_fitness\_value}")

# results\_text.yview\_moveto(1)

# ax.scatter(best\_solution[0], best\_solution[1], best\_fitness\_value, color='black', marker='x', s=60)

# results\_text.config(state=tk.DISABLED)

param\_frame2 = frame

# Параметры задачи

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

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

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

ttk.Label(param\_frame2, text="Задержка", font=("Helvetica", 10)).grid(row=6, 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)

delay=tk.DoubleVar(value=0.1)

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)

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

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

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

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

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

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

delay\_entry.grid(row=6, 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)).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)

ttk.Label(param\_frame2, text="X интервал (min)", font=("Helvetica", 10)).grid(row=11, column=0)

ttk.Label(param\_frame2, text="X интервал (max)", font=("Helvetica", 10)).grid(row=12, column=0)

ttk.Label(param\_frame2, text="Y интервал (min)", font=("Helvetica", 10)).grid(row=13, column=0)

ttk.Label(param\_frame2, text="Y интервал (max)", font=("Helvetica", 10)).grid(row=14, column=0)

ttk.Label(param\_frame2, text="Ось X интервал", font=("Helvetica", 10)).grid(row=16, column=0)

ttk.Label(param\_frame2, text="Ось Y интервал", font=("Helvetica", 10)).grid(row=17, column=0)

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

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

x\_interval\_min = tk.DoubleVar(value=-4)

x\_interval\_max = tk.DoubleVar(value=4)

y\_interval\_min = tk.DoubleVar(value=-4)

y\_interval\_max = tk.DoubleVar(value=4)

x\_axis\_interval = tk.IntVar(value=2)

y\_axis\_interval = tk.IntVar(value=2)

x\_interval\_min\_entry = ttk.Entry(param\_frame2, textvariable=x\_interval\_min)

x\_interval\_max\_entry = ttk.Entry(param\_frame2, textvariable=x\_interval\_max)

y\_interval\_min\_entry = ttk.Entry(param\_frame2, textvariable=y\_interval\_min)

y\_interval\_max\_entry = ttk.Entry(param\_frame2, textvariable=y\_interval\_max)

x\_axis\_interval\_entry = ttk.Entry(param\_frame2, textvariable=x\_axis\_interval)

y\_axis\_interval\_entry = ttk.Entry(param\_frame2, textvariable=y\_axis\_interval)

x\_interval\_min\_entry.grid(row=11, column=1)

x\_interval\_max\_entry.grid(row=12, column=1)

y\_interval\_min\_entry.grid(row=13, column=1)

y\_interval\_max\_entry.grid(row=14, column=1)

x\_axis\_interval\_entry.grid(row=16, column=1)

y\_axis\_interval\_entry.grid(row=17, column=1)

# 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)).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)