МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий

ЛАБОРАТОРНАЯ РАБОТА №4

Дисциплина: Методы поисковой оптимизации

Тема: Алгоритм роя частиц

Работу выполнили: _____ Парфинцов. Е. А.

_____ Татарян. Е. В.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль): Математическое и программное обеспечение компьютерных технологий

Преподаватель: _____ Полупанова Е. Е.

Краснодар
2024

Цель работы: разработать алгоритм роя частиц оптимизации функции Растригина.

Ход работы

Идея алгоритма роя частиц была частично заимствована из исследований поведения скоплений животных. Модель была немного упрощена и были добавлены элементы поведения толпы людей.

Блок-схема алгоритма представлена на рисунке 1.

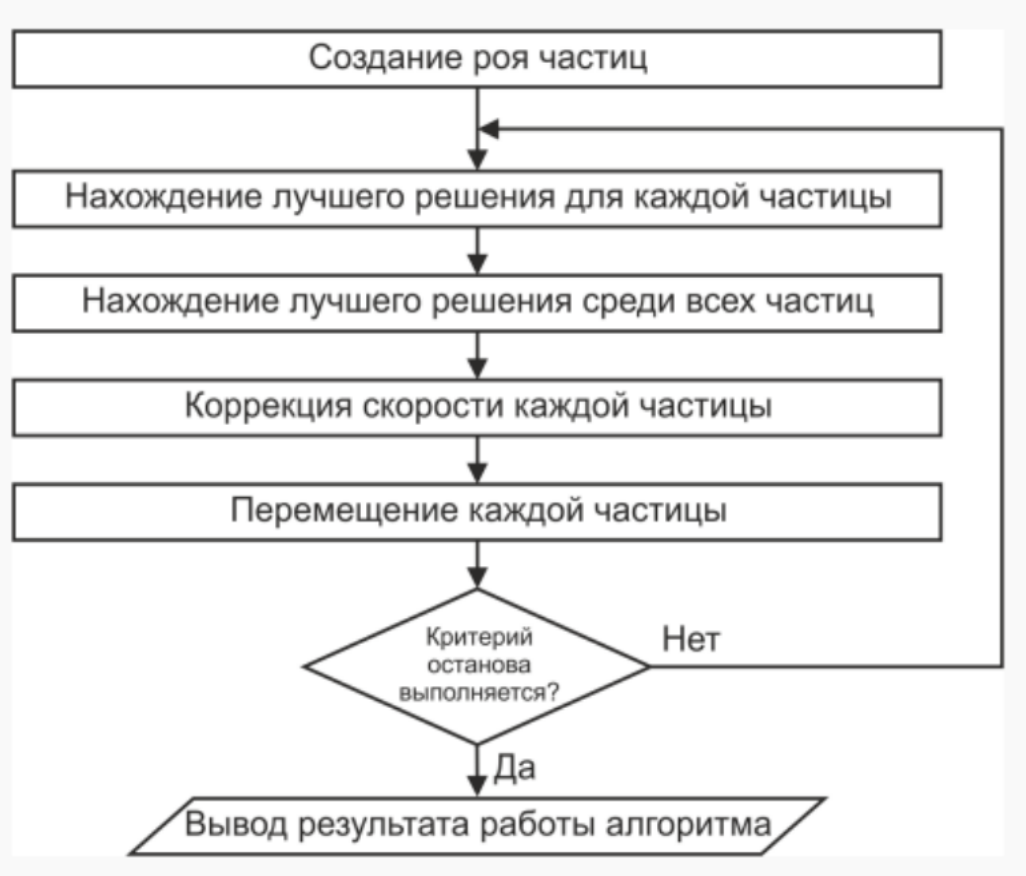


Рис. 1 Блок-схема алгоритма.

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 (и где) лучшее значение целевой функции она лично нашла, а также каждая частица знает где расположена точка, являющаяся лучшей среди всех точек, которые разведали частицы. На

каждой итерации частицы корректируют свою скорость (модуль и направление), чтобы, с одной стороны, быть поближе к лучшей точке, которую частица нашла сама (авторы алгоритма назвали этот аспект поведения "ностальгией"), и, в то же время, приблизиться к точке, которая в данный момент является глобально лучшей. Через некоторое количество итераций частицы должны собраться вблизи лучшей точки, хотя возможно, что часть частиц останется где-то в относительно неплохом локальном экстремуме, но главное, чтобы хотя бы одна частица оказалась вблизи глобального экстремума.

Самое интересное в алгоритме — это коррекция скорости, именно от этого шага зависит сходимость алгоритма. Коррекция скорости выглядит следующим образом:

$$v_{i,t+1} = v_{i,t} + \varphi_p r_p(p_i - x_{i,t}) + \varphi_g r_g(g_i - x_{i,t})$$

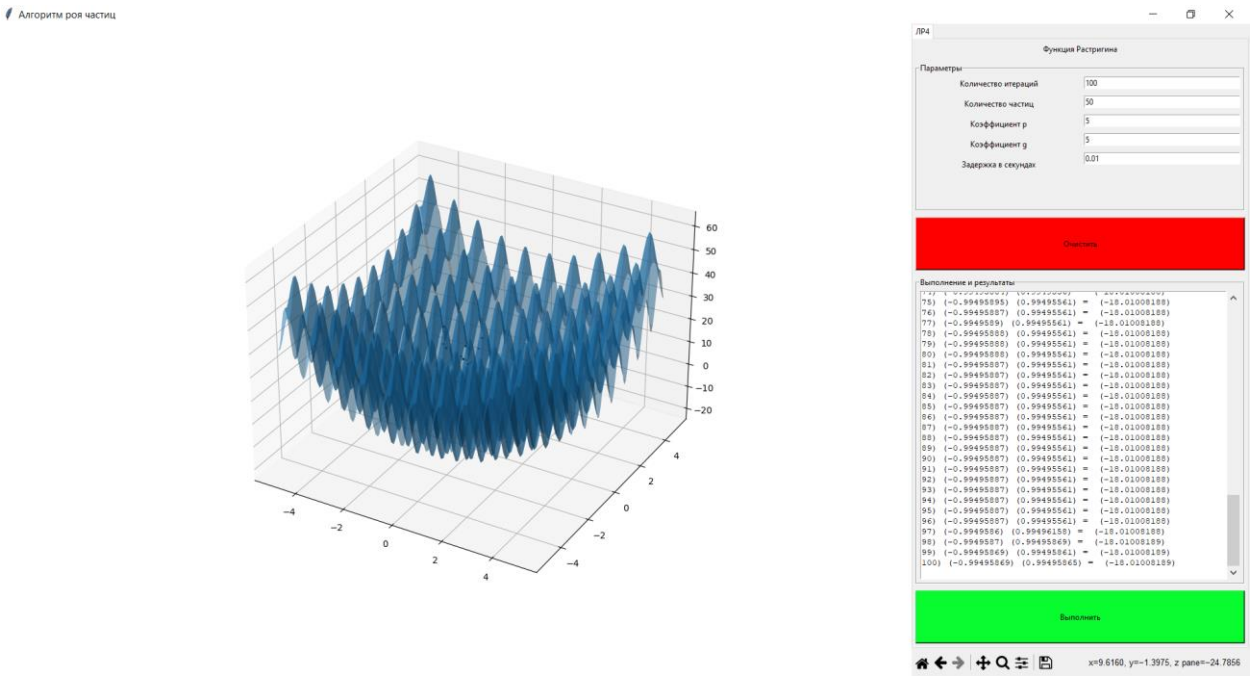
Здесь $v_{i,t}$ – i-я компонента скорости при t-ой итерации алгоритма $x_{i,t}$ – i-я координата частицы при t-ой итерации алгоритма p_i – i-я координата лучшего решения, найденного частицей g_i – i-я координата лучшего решения, найденного всеми частицами r_p, r_g – случайные числа в интервале (0, 1) φ_p, φ_g – весовые коэффициенты, которые надо подбирать под конкретную задачу.

Затем корректируем текущую координату каждой частицы:

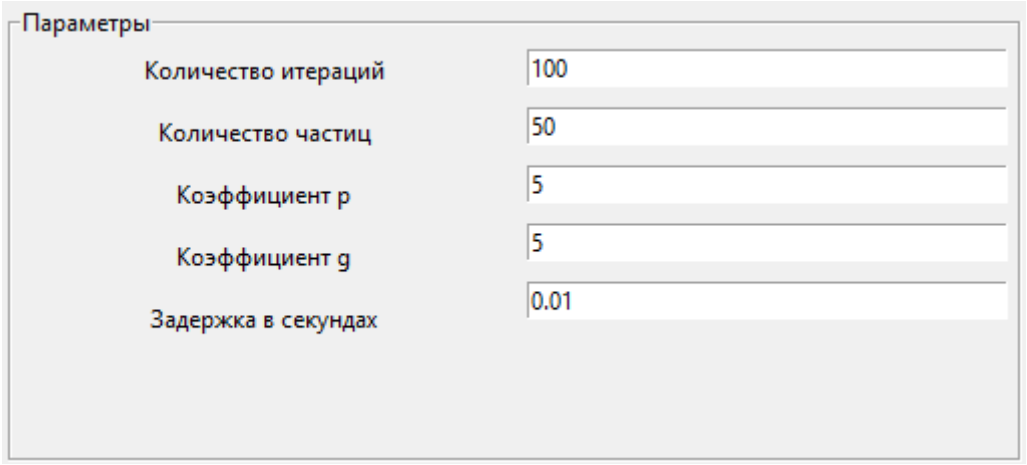
$$x_{i,t+1} = x_{i,t} + v_{i,t+1}$$

После этого рассчитываем 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 проверяем, не нашли ли мы новую глобально лучшую точку, и, если нашли, запоминаем ее координаты и значение целевой функции в ней.

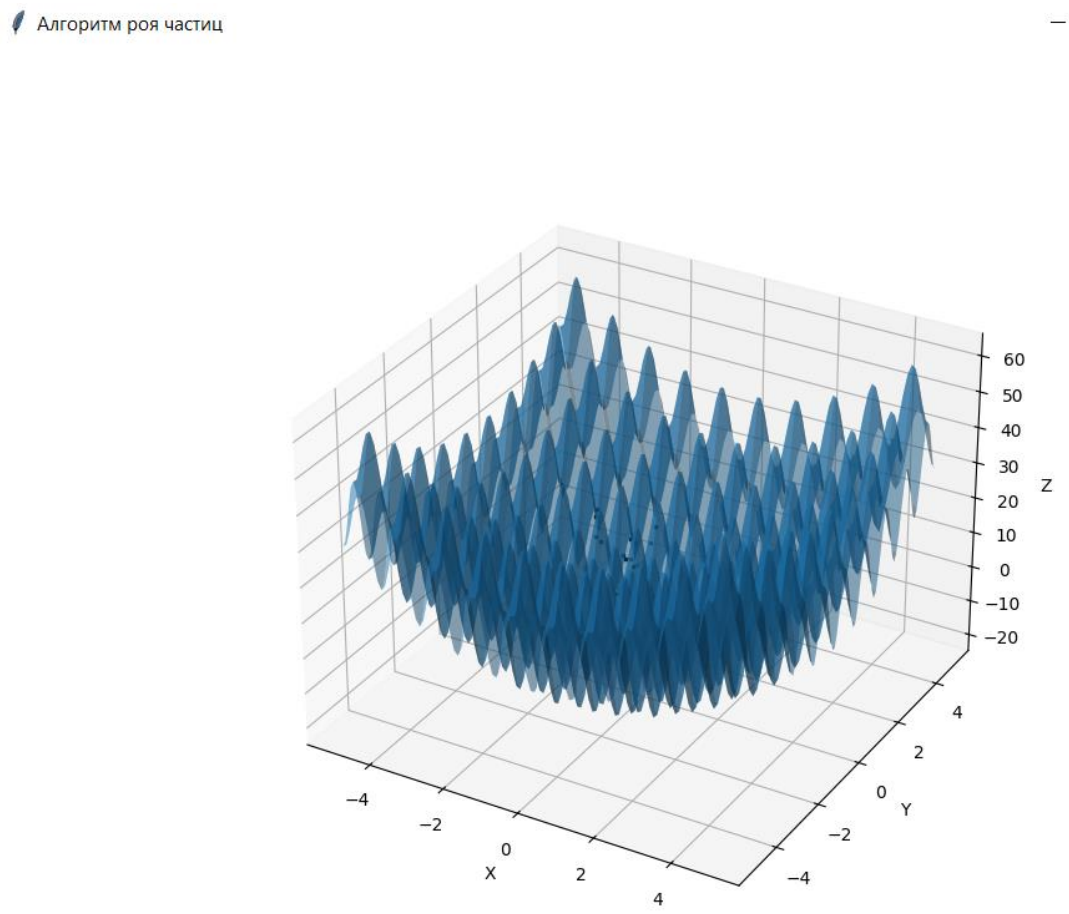
Реализуем данный алгоритм на языке Python (листинги 1–2):



Скриншот 1. Внешний вид приложения.



Скриншот 2. Параметры алгоритма.



Скриншот 3. Вид графика функции Растригина.

```
100) (-0.99495869) (0.99495865) =  (-18.01008189)
```

Скриншот 3. Результат работы

Листинг 1. (Шаги алгоритма)

```
import random
import numpy as np
from operator import itemgetter

class PSO:
    # В данном конструкторе инициализируем параметры алгоритма:
    # оптимизируемую функцию, кол-во частиц в стае, диапазоны начальных позиций
частиц по Ox и Oy,
    # а также коэффициенты для влияния на движение частиц.
    def __init__(self, func, population, position_x, position_y, fi_p, fi_g):
        self.func = func
        self.population = population

        self.pos_x = float(position_x)
        self.pos_y = float(position_y)

        # Проверяем, что fi_p + fi_g > 4, иначе срабатывает исключение.
        assert fi_p + fi_g > 4, "Сумма коэффициентов должна быть > 4"
        self.fi_p = fi_p
        self.fi_g = fi_g

        # Вычисляем параметр xi, который используется при обновлении скорости
частиц по формуле.
        self.xi = 2 / (np.abs(2 - (fi_p + fi_g) - np.sqrt((fi_p + fi_g) ** 2 - 4
* (fi_p + fi_g))))

        # Инициализируется стартовая популяция частиц particles, каждая из
которых представлена как список [x, y, fitness],
        # где x и y - начальные координаты частиц, а fitness - значение функции
func в этих координатах.
        self.particles = [[random.uniform(-self.pos_x, self.pos_x),
random.uniform(-self.pos_y, self.pos_y), 0.0]
                          for _ in range(self.population)]
        for i in self.particles:
            i[2] = self.func(i[0], i[1])

        # Создается копия популяции nostalgia, использующаяся для хранения
"лучших" позиций частиц.
        self.nostalgia = self.particles.copy()

        # Инициализируется массив velocity, который представляет скорость каждой
частицы (изначально все скорости установлены в 0).
        self.velocity = [[0.0 for _ in range(2)] for _ in range(self.population)]

        # Находится начальное лучшее решение generation_best, выбирая частицу с
минимальным значением fitness из текущей популяции.
        self.generation_best = min(self.particles, key=itemgetter(2))

    # update_velocity используется для обновления скорости частиц.
```

```
    def update_velocity(self, velocity, particle, point_best) -> list:
        new_vel = list()
        for i in range(2):
            r1 = random.random()
            r2 = random.random()

            new_vel.append(self.xi * (velocity[i] + self.fi_p * r1 *
(point_best[i] - particle[i]) + self.fi_g * r2 * (
                    self.generation_best[i] - particle[i])))
        return new_vel

    # update_position используется для обновления позиции частиц.
    def update_position(self, velocity, particle):
        x = particle[0] + velocity[0]
        y = particle[1] + velocity[1]

        return [x, y, self.func(x, y)]

    # next_iteration обновляет скорость и позицию каждой частицы и сравнивает их
со значениями в nostalgia,
    # затем обновляет generation_best, который будет содержать координаты
наилучшей позиции.
    def next_iteration(self):
        for i in range(self.population):

            if self.nostalgia[i][2] < self.particles[i][2]:
                point_best = self.nostalgia[i]
            else:
                self.nostalgia[i] = self.particles[i]
                point_best = self.particles[i]

            self.velocity[i] = PSO.update_velocity(self, self.velocity[i],
self.particles[i], point_best)
            self.particles[i] = PSO.update_position(self, self.velocity[i],
self.particles[i])

        self.generation_best = min(self.particles, key=itemgetter(2))
```

Листинг 2. (Приложение и запуск алгоритма)

```
import tkinter
import time
from tkinter import *
from tkinter import scrolledtext, messagebox
from tkinter.ttk import Notebook
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
from pso import PSO
from functions import *

def main():
    window = Tk()

    window.geometry("1000x800")

    window.title("Алгоритм роя частиц")

    fig = plt.figure(figsize=(14, 14))
    fig.add_subplot(projection='3d')

    canvas = FigureCanvasTkAgg(fig, master=window)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)

    toolbar = NavigationToolbar2Tk(canvas, window)
    toolbar.update()
    canvas.get_tk_widget().pack(side=tkinter.LEFT, fill=tkinter.BOTH)


    tab_control = Notebook(window)

    # Лаба 4
    def draw_lab_4():
        fig.clf()

        #x, y, z = make_data_lab_3()
        px = 5.0
        py = 5.0
        x, y, z = make_data_rastrigin(px, py)

        iter_number = int(txt_1_tab_4.get())
        particle_number = int(txt_2_tab_4.get())
        fi_p = float(txt_4_tab_4.get())
        fi_g = float(txt_5_tab_4.get())
        delay = txt_6_tab_4.get()

        ax = fig.add_subplot(projection='3d')
        ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
```

```
        canvas.draw()

        psa_obj = PSO(rastrigin_2, particle_number, px, py, fi_p, fi_g)

        for particle in psa_obj.particles:
            ax.scatter(particle[0], particle[1], particle[2], c="black", s=1,
marker="s")

        ax.scatter(psa_obj.generation_best[0], psa_obj.generation_best[1],
psa_obj.generation_best[2], c="red")
        canvas.draw()
        window.update()

        # Эти 4 строки ниже отвечают за удаление точки
        fig.clf()
        ax = fig.add_subplot(projection='3d')
        ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
        canvas.draw()

        for i in range(iter_number):
            psa_obj.next_iteration()
            for particle in psa_obj.particles:
                ax.scatter(particle[0], particle[1], particle[2], c="black", s=1,
marker="s")

            ax.scatter(psa_obj.generation_best[0], psa_obj.generation_best[1],
psa_obj.generation_best[2], c="red")

            txt_tab_4.insert(INSERT,
                             f"{i + 1}) ({round(psa_obj.generation_best[0], 8)})"
                             f" ({round(psa_obj.generation_best[1], 8)}) = "
                             f" ({round(psa_obj.generation_best[2], 8)})\n")

            canvas.draw()
            window.update()
            time.sleep(float(delay))

            fig.clf()
            ax = fig.add_subplot(projection='3d')
            ax.plot_surface(x, y, z, rstride=5, cstride=5, alpha=0.5)
            canvas.draw()

        for particle in psa_obj.particles:
            ax.scatter(particle[0], particle[1], particle[2], c="black", s=1,
marker="s")

        ax.scatter(psa_obj.generation_best[0], psa_obj.generation_best[1],
psa_obj.generation_best[2], c="red")

        canvas.draw()
        ax.set_xlabel('X')
```

```
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        window.update()

        messagebox.showinfo('Уведомление', 'Готово')

    def delete_lab_4():
        txt_tab_4.delete(1.0, END)

    tab_4 = Frame(tab_control)
    tab_control.add(tab_4, text="ЛР4")

    main_f_tab_4 = LabelFrame(tab_4, text="Параметры")
    left_f_tab_4 = Frame(main_f_tab_4)
    right_f_tab_4 = Frame(main_f_tab_4)
    txt_f_tab_4 = LabelFrame(tab_4, text="Выполнение и результаты")

    lbl_1_tab_4 = Label(left_f_tab_4, text="Количество итераций")
    lbl_2_tab_4 = Label(left_f_tab_4, text="Количество частиц")
    lbl_4_tab_4 = Label(left_f_tab_4, text="Коэффициент g")
    lbl_5_tab_4 = Label(left_f_tab_4, text="Задержка в секундах")
    lbl_6_tab_4 = Label(tab_4, text="Функция Растригина")
    lbl_7_tab_4 = Label(left_f_tab_4, text="Коэффициент p")

    txt_1_tab_4 = Entry(right_f_tab_4)
    txt_1_tab_4.insert(0, "100")
    txt_2_tab_4 = Entry(right_f_tab_4)
    txt_2_tab_4.insert(0, "50")
    txt_4_tab_4 = Entry(right_f_tab_4)
    txt_4_tab_4.insert(0, "5")
    txt_5_tab_4 = Entry(right_f_tab_4)
    txt_5_tab_4.insert(0, "5")
    txt_6_tab_4 = Entry(right_f_tab_4)
    txt_6_tab_4.insert(0, "0.01")

    txt_tab_4 = scrolledtext.ScrolledText(txt_f_tab_4)
    btn_del_tab_4 = Button(tab_4, text="Очистить",
background="red",command=delete_lab_4)
    btn_tab_4 = Button(tab_4, text="Выполнить", foreground="black",
background="#08fc30", command=draw_lab_4)

    lbl_6_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    main_f_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH, expand=True)
    left_f_tab_4.pack(side=LEFT, fill=BOTH, expand=True)
    right_f_tab_4.pack(side=RIGHT, fill=BOTH, expand=True)

    lbl_1_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_2_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_7_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_4_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    lbl_5_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
```

```
    txt_1_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_2_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_4_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_5_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)
    txt_6_tab_4.pack(side=TOP, padx=5, pady=5, fill=BOTH)

    txt_tab_4.pack(padx=5, pady=5, fill=BOTH, expand=True)

    btn_tab_4.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)
    txt_f_tab_4.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)
    btn_del_tab_4.pack(side=BOTTOM, padx=5, pady=5, fill=BOTH, expand=True)

    tab_control.pack(side=RIGHT, fill=BOTH, expand=True)
    window.mainloop()

if __name__ == '__main__':
    main()
```