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 №4**

**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](#_Toc121176630)

[Алгоритм 3](#_Toc121176631)

[Результат работы программы 4](#_Toc121176632)

[Листинг 6](#_Toc121176633)

Тема работы: Алгоритм роя частиц.

# Задание

Необходимо разработать программу оптимизации функции сферы.

*.*

Ее глобальный (и единственный) минимум расположен в точке . Значение функции в этой точке равно 0.

# Алгоритм

Блок-схема алгоритма представлена на рисунке 1.

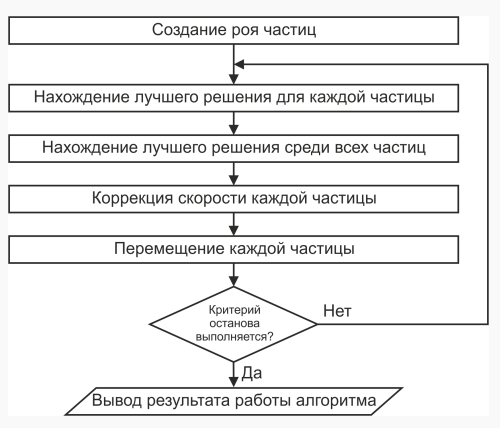


Рисунок 1 – Блок-схема алгоритма

Текущее состояние частицы характеризуется координатами в пространстве решений, а также вектором скорости перемещения, который вычисляется по формуле:

,

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

Координаты частицы и вектор скорости выбираются случайным образом на этапе инициализации. Кроме того, каждая частица хранит координаты лучшего из найденных ей решений, а также лучшее из пройденных всеми частицами решений.

На каждой итерации алгоритма направление вектора скорости каждой из частиц изменяется в соответствие со сведениями о найденных оптимумах.

После вычисления направления вектора частица перемещается в точку по формуле:

В случае необходимости обновляются значения лучших точек для каждой частицы и для всех частиц в целом. После этого цикл повторяется.

# Результат работы программы

Вся работа программы представлена на рисунках 2-3 ниже.

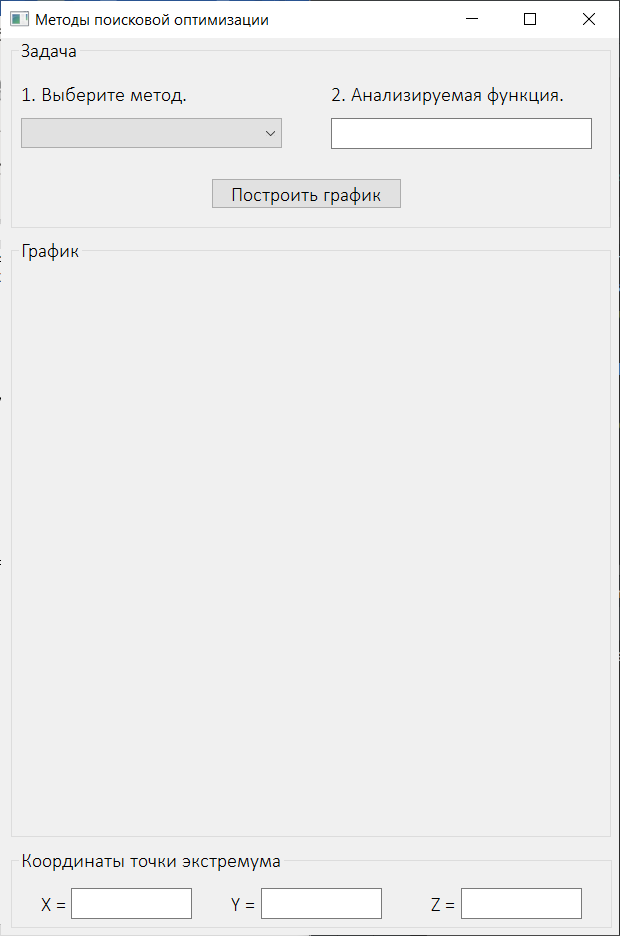


Рисунок 2 – Начальный вид программы при запуске

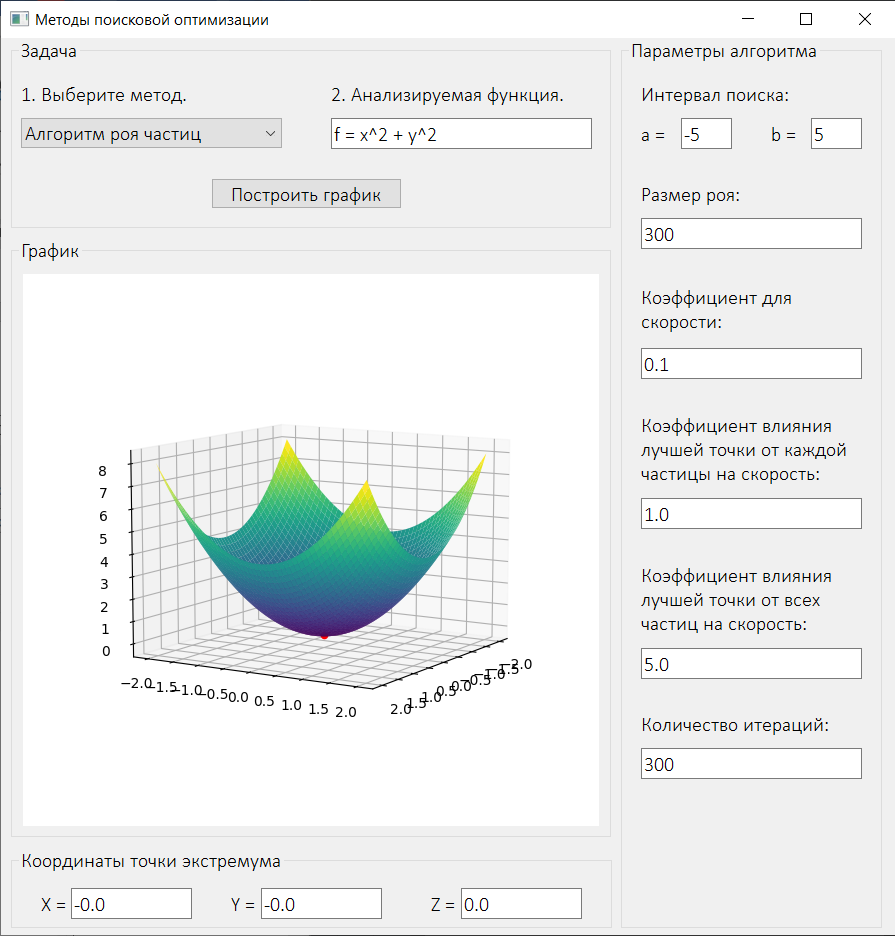


Рисунок 3 – Вид программы при выборе метода градиентного спуска с постоянным шагом

# Листинг

import numpy

import numpy.random

class Swarm:

def \_\_init\_\_(self, swarm\_size, min\_values, max\_values, current\_velocity\_ratio, local\_velocity\_ratio, global\_velocity\_ratio):

self.swarm\_size = swarm\_size

self.min\_values = numpy.array(min\_values[:])

self.max\_values = numpy.array(max\_values[:])

self.current\_velocity\_ratio = current\_velocity\_ratio

self.local\_velocity\_ratio = local\_velocity\_ratio

self.global\_velocity\_ratio = global\_velocity\_ratio

self.global\_best\_final\_func = None

self.global\_best\_position = None

self.swarm = self.create\_swarm()

def create\_swarm(self):

return [Particle(self) for i in range(self.swarm\_size)]

def next\_iteration(self):

for particle in self.swarm:

particle.next\_iteration(self)

def get\_min\_values(self):

return self.min\_values

def get\_max\_values(self):

return self.max\_values

def get\_current\_velocity\_ratio(self):

return self.current\_velocity\_ratio

def get\_local\_velocity\_ratio(self):

return self.local\_velocity\_ratio

def get\_global\_velocity\_ratio(self):

return self.global\_velocity\_ratio

def get\_global\_best\_position(self):

return self.global\_best\_position

def get\_global\_best\_final\_func(self):

return self.global\_best\_final\_func

def get\_final\_func(self, position):

final\_func = self.final\_func(position)

if self.global\_best\_final\_func is None or final\_func < self.global\_best\_final\_func:

self.global\_best\_final\_func = final\_func

self.global\_best\_position = position[:]

return final\_func

def final\_func (self, position):

penalty = self.get\_penalty (position, 10000.0)

final\_func = sum(position \* position)

return final\_func + penalty

def dimension(self):

return len(self.min\_values)

def get\_penalty(self, position, ratio):

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

for coord, min\_val in zip(position, self.min\_values)

if coord < min\_val])

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

for coord, max\_val in zip(position, self.max\_values)

if coord > max\_val])

return penalty1 + penalty2

class Particle:

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

self.current\_position = self.get\_init\_position(swarm)

self.local\_best\_position = self.current\_position[:]

self.local\_best\_final\_func = swarm.get\_final\_func(self.current\_position)

self.velocity = self.get\_init\_velocity(swarm)

def get\_position(self):

return self.current\_position

def get\_velocity(self):

return self.velocity

@staticmethod

def get\_init\_position(swarm):

return numpy.random.rand(swarm.dimension()) \* (swarm.get\_max\_values() - swarm.get\_min\_values()) + swarm.get\_min\_values()

@staticmethod

def get\_init\_velocity(swarm):

min\_val = -(swarm.get\_max\_values() - swarm.get\_min\_values())

max\_val = (swarm.get\_max\_values() - swarm.get\_min\_values())

return numpy.random.rand(swarm.dimension()) \* (max\_val - min\_val) + min\_val

def next\_iteration(self, swarm):

rnd\_current\_best\_position = numpy.random.rand(swarm.dimension())

rnd\_global\_best\_position = numpy.random.rand(swarm.dimension())

velo\_ratio = swarm.get\_local\_velocity\_ratio() + swarm.get\_global\_velocity\_ratio()

common\_ratio = (2.0 \* swarm.get\_current\_velocity\_ratio() /

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

new\_velocity\_part1 = common\_ratio \* self.velocity

new\_velocity\_part2 = (common\_ratio \*

swarm.get\_local\_velocity\_ratio() \*

rnd\_current\_best\_position \*

(self.local\_best\_position - self.current\_position))

new\_velocity\_part3 = (common\_ratio \*

swarm.get\_global\_velocity\_ratio() \*

rnd\_global\_best\_position \*

(swarm.get\_global\_best\_position() - self.current\_position))

self.velocity = new\_velocity\_part1 + new\_velocity\_part2 + new\_velocity\_part3

self.current\_position += self.velocity

final\_func = swarm.get\_final\_func(self.current\_position)

if final\_func < self.local\_best\_final\_func:

self.local\_best\_position = self.current\_position[:]

self.local\_best\_final\_func = final\_func

class ParticleSwarmMethod:

@staticmethod

def particle\_swarm\_method(swarm\_size, min\_values, max\_values, current\_velocity\_ratio, local\_velocity\_ratio, global\_velocity\_ratio, iterations):

swarm = Swarm(swarm\_size, min\_values, max\_values, current\_velocity\_ratio, local\_velocity\_ratio, global\_velocity\_ratio)

for n in range(iterations):

swarm.next\_iteration()

min\_point = swarm.get\_global\_best\_position().tolist()

return min\_point

# if \_\_name\_\_ == "\_\_main\_\_":

# psw = ParticleSwarmMethod()

# print(psw.particle\_swarm\_method(200, numpy.array ([-5] \* 2), numpy.array ([5] \* 2), 0.1, 1.0, 5.0, 300))