

| МИНОБРНАУКИ РОССИИ |
| --- |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы:ИКБО-42-23 \_\_\_Голев С.С.\_\_\_ *(Ф. И.О.студента)*

Преподаватель \_\_Железняк Л.М.\_\_ *(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_pgf9r1stsb9d)

[3 РОЕВОЙ АЛГОРИТМ 4](#_qzqtm92707g1)

[3.1 Цель и задачи практической работы 4](#_fq21lbpksb4y)

[3.2 Постановка задачи 5](#)

[3.3 Ручной расчёт 6](#)

[3.4 Результат работы метода отжига 8](#_rxn9v8bp34u5)

[3.5 Результат работы нахождения минимума 9](#)

[ЗАКЛЮЧЕНИЕ 10](#_fil1vc9v6soa)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 11](#_la6yu080q6h9)

[ПРИЛОЖЕНИЯ 12](#_tx7j3zan39q)

# 

# ВВЕДЕНИЕ

Роевые алгоритмы относятся к классу эвристических методов оптимизации, вдохновленных коллективным поведением живых организмов. Основная идея заключается в моделировании взаимодействия между частицами или агентами, которые совместно ищут оптимальное решение. В работе рассматривается применение роевого алгоритма для нахождения минимума функции и решения задачи коммивояжёра. Такой подход основан на обмене информацией между частицами, что позволяет эффективно исследовать пространство решений. Алгоритм обеспечивает баланс между поиском новых областей и уточнением найденных решений. В результате роевые методы демонстрируют высокую производительность и способность избегать локальных минимумов.

# 

# 3 РОЕВОЙ АЛГОРИТМ

# 3.1 Цель и задачи практической работы

Целью практической работы является изучение принципов роевого алгоритма и его применение для решения задач оптимизации. В рамках работы необходимо освоить механику обновления положения и скорости частиц, а также влияние обмена информацией между ними на поиск оптимального решения.

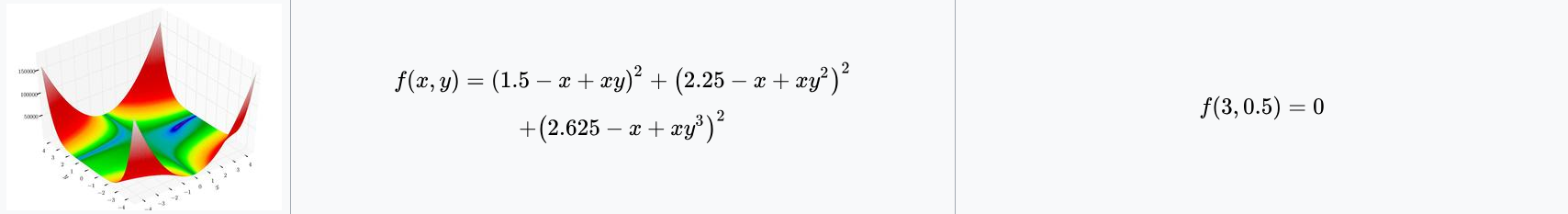
Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить ручной расчёт одной итерации роевого алгоритма для заданной функции, определяя новые координаты частиц и значения функции;
2. Реализовать роевой алгоритм на языке Python для автоматического поиска минимума функции;
3. Применить алгоритм для решения задачи коммивояжёра и минимизации суммарного расстояния маршрута;
4. Проанализировать влияние параметров алгоритма (скорости, веса инерции, коэффициенты притяжения) на качество и скорость сходимости;
5. Сравнить эффективность результатов ручного расчёта и программной реализации, выявив преимущества алгоритма для задач оптимизации.

# 3.2 Постановка задачи

В рамках практической работы необходимо реализовать роевый алгоритм вручную и кодово, на примере нахождения минимум функции.

Как тестовая функция была выбрана функция Била.



**Рисунок 1 –** Функция Била

# 

# 3.3 Ручной расчёт

Выполним ручной расчёт одной итерации, для функции Била диапазон значений для *x* и *y* равен [–4.5; 4.5].

Скорость будет рассчитываться по формуле 3.1.

| vi(t+1) = vi(t) + c1\*r1\*(ybesti – xi) + c2 \* r2 \* (ybest – xi) | (3.1) |
| --- | --- |

Где ybesti лучшее значение данной точки, ybest лучшее значение точки всего роя. с1 и с2 константы, r1 и r2 случайные числа из диапазона [0;1].

В нашем случае за константы c возьмём значение 2, а для констант r возьмём значение 0,3 и 0,7 соответственно.

Проинициализируем 4 единицы роя.

*Таблица 3.1 – Начальные значения роя*

|  | x | y | f(x,y) |
| --- | --- | --- | --- |
| 1 | -2.9 | 0.8 | 105.28 |
| 2 | 0.17 | -3.16 | 25.56 |
| 3 | -1.93 | -3.29 | 5764.5 |
| 4 | -4.02 | -1.25 | 319.49 |

В данной таблице лучшее значение имеет элемент 2, следовательно ybest будет иметь значение 2-ого элемента.

Далее вычисляем скорость для каждой координаты частицы, после чего пересчитаем новые координаты точек.

*Таблица 3.2 – Значения скорости роя и новые координаты точек*

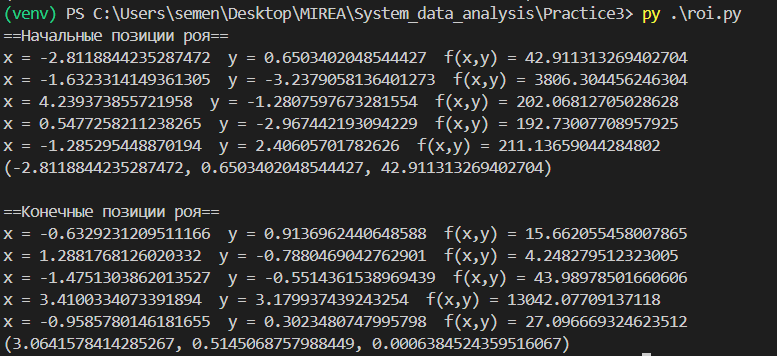
|  | Vx | Vy | x | y |
| --- | --- | --- | --- | --- |
| 1 | 4,30 | -5,54 | 1,40 | -4,44 |
| 2 | 0 | 0 | 0,17 | -3,16 |
| 3 | 2,94 | 0,18 | 1,01 | -3,11 |
| 4 | 5,87 | -2,67 | 1,85 | -3,92 |

Далее считаем значение функций, для новых координат, находим новый глобальный минимум и локальный минимум для каждой точки, после чего повторяем итерацию с вычислением скорости.

# 

# 3.4 Результат работы метода отжига

Реализуем нахождение минимума функции с помощью роевого алгоритма, количество частиц: 5, количество итераций: 100, выполним реализацию на языке Python. Реализация представлена в приложении В.



**Рисунок 2 – Пример выполнения роевого алгоритма**

# 

# 3.5 Результат работы нахождения минимума

В ходе практической работы была выполнена одна итерация ручного расчёта роевого алгоритма для двух частиц.

Данный расчёт демонстрирует работу роевого алгоритма: частицы корректируют свои скорости и позиции, ориентируясь на личный и глобальный опыт, постепенно приближаясь к оптимальному значению.

Далее была выполнена кодовая реализация роевого алгоритма, которая позволила:

1. Автоматически находить минимум функции в многомерном пространстве.
2. Применять алгоритм для решения задачи коммивояжёра, минимизируя суммарное расстояние маршрута.
3. Анализировать влияние параметров алгоритма (веса инерции, коэффициенты притяжения) на скорость сходимости и точность результата.

В результате работы показано, что роевой алгоритм является эффективным методом как для непрерывной, так и для комбинаторной оптимизации, обеспечивая скоординированный поиск оптимальных решений.

# 

# ЗАКЛЮЧЕНИЕ

В ходе работы был реализован роевой алгоритм для решения задач оптимизации различного типа. Эксперименты показали, что данный метод способен эффективно приближаться к глобальному минимуму и находить качественные решения задачи коммивояжёра. Важным преимуществом алгоритма является способность адаптироваться к структуре задачи за счёт коллективного обмена информацией. Роевое поведение частиц обеспечивает устойчивость к случайным возмущениям и ускоряет сходимость. Полученные результаты подтвердили универсальность и надёжность роевого подхода. Таким образом, использование роевого алгоритма представляет собой эффективный инструмент для решения сложных задач оптимизации.

# 

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: https://docs.python.org/3/ (дата обращения: 15.09.2025).
2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.
3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.
4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

# 

# ПРИЛОЖЕНИЯ

Приложение В – Код программы “Роевой алгоритм”

**Приложение А**

Код программы Онтологии

*Листинг В.1 — Основной алгоритм программы*

| import numpy as np  def f(x, y):  return (1.5 - x + x\*y)\*\*2 + (2.25 - x + x\*(y\*\*2))\*\*2 + (2.625 - x + x\*(y\*\*3))\*\*2  class Particle():  def \_\_init\_\_(self):  self.xi = []  self.yi = []  self.func = []  self.Vxi = []  self.Vyi = []    def find\_best(self):  minf = min(self.func)  for i in range(len(self.func)):  if minf == self.func[i]:  return self.xi[i], self.yi[i]  class Roi():  def \_\_init\_\_(self):  self.parts = []  self.glob\_best = ()  self.iteration = 0    self.c1, self.c2 = 2, 2  self.r1, self.r2 = np.random.uniform(0, 1), np.random.uniform(0, 1)  def create(self, count, minZ, maxZ):  for \_ in range (count):  part = Particle()    part.xi.append(np.random.uniform(minZ, maxZ))  part.yi.append(np.random.uniform(minZ, maxZ))  part.func.append(f(part.xi[0],part.yi[0]))  part.Vxi.append(np.random.uniform(minZ, maxZ))  part.Vyi.append(np.random.uniform(minZ, maxZ))    self.parts.append(part)  minf = min(obj.func for obj in self.parts)  for obj in self.parts:  self.glob\_best = (obj.xi[0], obj.yi[0], obj.func[0]) if minf == obj.func else self.glob\_best |
| --- |

*Листинг В.2 — Продолжение листинга В.1*

| def new\_v(self, n, xb, yb):  return (  self.parts[n].Vxi[self.iteration]  + self.c1 \* self.r1 \* (xb - self.parts[n].xi[self.iteration])  + self.c2 \* self.r2 \* (self.glob\_best[0] - self.parts[n].xi[self.iteration])  ),(  self.parts[n].Vyi[self.iteration]  + self.c1 \* self.r1 \* (yb - self.parts[n].yi[self.iteration])  + self.c2 \* self.r2 \* (self.glob\_best[1] - self.parts[n].yi[self.iteration])  )    def make\_iter(self, N):  for i in range(N):  xb, yb = self.parts[i].find\_best()  Vx, Vy = self.new\_v(i, xb, yb)    self.parts[i].Vxi.append(Vx)  self.parts[i].Vyi.append(Vy)    self.parts[i].xi.append(self.parts[i].xi[self.iteration] + Vx)  self.parts[i].yi.append(self.parts[i].yi[self.iteration] + Vy)  self.parts[i].func.append(f(self.parts[i].xi[self.iteration] + Vx, self.parts[i].yi[self.iteration] + Vy))    minf = min(min(obj.func) for obj in self.parts)  for obj in self.parts:  for i in range (len(obj.func)):  self.glob\_best = (obj.xi[i], obj.yi[i], obj.func[i]) if minf == obj.func[i] else self.glob\_best    self.iteration += 1  def print(self):  for obj in self.parts:  print(f'x = {obj.xi[self.iteration]} y = {obj.yi[self.iteration]} f(x,y) = {obj.func[self.iteration]}') |
| --- |

*Листинг В.3— Продолжение листинга В.2*

| if \_\_name\_\_ == '\_\_main\_\_':  obj = Roi()  N = 5  max\_iter = 100  obj.create(N, -4.5, 4.5)    print("==Начальные позиции роя==")  obj.print()  print(obj.glob\_best)  while obj.iteration != max\_iter:  obj.make\_iter(N)  print("\n==Конечные позиции роя==")  obj.print()  print(obj.glob\_best) |
| --- |