

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА** - **Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №6**

**«Электромагнитный алгоритм»**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы: ИКБО-04-22 Егоров Л.А. *(Ф. И.О.студента)*

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

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

Москва 2024

# СОДЕРЖАНИЕ

[Введение 3](#_Toc184212443)

[1 Электромагнитный алгоритм 4](#_Toc184212444)

[1.1 Описание алгоритма 4](#_Toc184212445)

[1.2 Постановка задачи 6](#_Toc184212446)

[1.3 Ручной расчёт алгоритма 7](#_Toc184212447)

[1.4 Программная реализация 11](#_Toc184212448)

[Заключение 12](#_Toc184212449)

[Список информационных источников 13](#_Toc184212450)

[Приложения 14](#_Toc184212451)

# ВВЕДЕНИЕ

Электромагнитный алгоритм предложен Бирбилом (I. Birbil) и Фангом (S.C. Fang) в 2003 году [2]. Этот алгоритм вдохновлен фундаментальными принципами электромагнетизма, а именно взаимодействием между электрическими зарядами и магнитными полями.

В алгоритме пространство поиска представляется как заполненное частицами, каждая из которых обладает определенным электрическим зарядом. Эти частицы взаимодействуют друг с другом посредством электромагнитных сил. Сила притяжения или отталкивания между двумя частицами зависит от их зарядов и расстояния между ними.

Каждого из агентов популяции в электромагнитном алгоритме интерпретируют как заряженную частицу, заряд которой пропорционален значению фитнесс-функции в той точке области поиска, в которой на данной итерации находится агент. Текущий заряд частиц популяции определяет суммарную силу, действующую на данную частицу со стороны других частиц, а также направление и величину её перемещений на текущей итерации. В соответствии с законами электростатики эта сила вычисляется путём векторного суммирования сил притяжения и отталкивания со стороны всех частиц популяции [2].

Электромагнитный алгоритм широко используется для решения различных задач оптимизации, таких как:

* задача коммивояжера;
* распределение ресурсов;
* планирование;
* сетевое проектирование.

# 1 Электромагнитный алгоритм

## 1.1 Описание алгоритма

Сначала происходит инициализация начальных параметров и зарядов – генерация точек в области поиска (количество точек задано и равно S), а также свободных параметров алгоритма. Каждая точка имеет координаты (1.1.1).

(1.1.1)

где — номер частицы;

— размерность векторов в задаче.

Затем начинается локальный поиск – для каждого заряда выполняется линейный стохастический поиск (1.1.2) в целях сбора локальной информации об окружении частиц [2, c.22-23].

(1.1.2)

где – случайное целое число, равное -1 или 1;

– случайное число от 0 до 1 из равномерного распределения;

– свободный параметр алгоритма;

- границы рассматриваемой области поиска.

При этом, изменённая координата принимается только в том случае, если её изменение дало улучшение значения целевой функции, иначе происходит переход к следующей итерации локального поиска.

Как правило, свободный параметр задаётся близким к нулю (порядка , поэтому локальный поиск служит только для незначительного улучшения положения точек в пространстве.

После осуществления локального поиска вычисляется лучшая точка, т.е. точка, где достигается минимальное среди всех значение целевой функции.

Затем для каждой частицы вычисляется её заряд по Формуле 1.1.3.

(1.1.3)

где – значение целевой функции в текущей точке;

– значение целевой функции в лучшей точке.

На основе посчитанных зарядов происходит вычисление силы отталкивания и притяжения между частицами (1.1.4). Частица i отталкивается от частицы j, если значение функции у частицы i лучше, чем значение функции у частицы j, и наоборот, притягивается, если значение функции хуже.

(1.1.4)

Таким образом, лучшая частица на данной итерации притянет к себе все остальные. Далее выполняется перемещение частиц, высчитываемое на основе электромагнитных сил (1.1.5).

(1.1.5)

где – случайное число от 0 до 1 из равномерного распределения;

– нормированный вектор силы;

– вектор скорости, компоненты которого рассчитываются по Формуле (1.1.6).

(1.1.6)

где - границы области поиска.

Важно отметить, что лучшая частица должна остаться на своём месте, поэтому она в данной итерации не передвигается, а только притягивает к себе другие частицы.

После осуществления перемещения частиц происходит переход к следующей итерации. Точкой останова алгоритма является достижение максимального числа итераций.

## 1.2 Постановка задачи

Цель работы: реализовать электромагнитный алгоритм для нахождения оптимального значения функции.

Поставлены следующие задачи:

* изучить электромагнитный алгоритм;
* выбрать тестовую функцию для оптимизации (нахождение глобального минимума);
* произвести ручной расчёт одной итерации алгоритма;
* разработать программную реализацию электромагнитного алгоритма для задачи минимизации функции.

Выбранная функция для оптимизации: функция Растригина (1.2.1). Глобальный минимум функции достигается в точке (0;0) и равен 0, при этом, в остальных локальных минимумах значение функции больше нуля. Функция рассматривается на области .

(1.2.1)

## 1.3 Ручной расчёт алгоритма

Выбранная функция: функция Растригина от двух переменных. Её формула представлена Формулой 1.2.1.

На Рисунке 1.3.1 представлен график этой функции.



Рисунок 1.3.1 – График функции Растригина

Количество частиц, используемых для ручного расчёта: 4.

Создано 4 частицы со следующими координатами:

После выполнения локального поиска точки имеют следующие координаты:

При этом, значение равно 34.224 в точке (1.993, 3.414).

Далее приведены расчёты значения заряда для каждой из частиц по Формуле 1.1.3:

При этом, заряд у лучшей частицы будет равен единице, поскольку аргумент в экспоненте будет равен нулю.

Затем вычисляются значение силы для каждой частицы по Формуле 1.1.4:

И наконец, позиции частиц изменяются по Формулам 1.1.5-1.1.6. Частица не передвигается, т.к. она имеет лучшее значение

Рассчитаем смещение для 1-й частицы. Сгенерировано случайное число

Нормированный вектор силы у 1-й частицы:

Рассчитаны компоненты вектора скорости для 1-й частицы:

Тогда частица сместится на следующую позицию:

Рассчитаем смещение для 2-й частицы. Сгенерировано случайное число

Нормированный вектор силы у 2-й частицы:

Рассчитаны компоненты вектора скорости для 2-й частицы:

Тогда частица сместится на следующую позицию:

Рассчитаем смещение для 4-й частицы. Сгенерировано случайное число

Нормированный вектор силы у 4-й частицы:

Рассчитаны компоненты вектора скорости для 4-й частицы:

Тогда частица сместится на следующую позицию:

После этого происходит переход на новую итерацию.

## 1.4 Программная реализация

Для реализации расчётов электромагнитного алгоритма написан программный код на языке Python.

В программной реализации зафиксированы следующие параметры:

* количество частиц: 20;
* количество итераций локального поиска: 10;
* количество общих итераций: 100;
* .

Код реализации электромагнитного алгоритма для нахождения оптимального значения функции представлен в Листинге А.1.

На Рисунке 1.4.1 представлен результат выполнения программы для нахождения оптимального значения функции – график зависимости минимального оптимального решения от номера итерации.

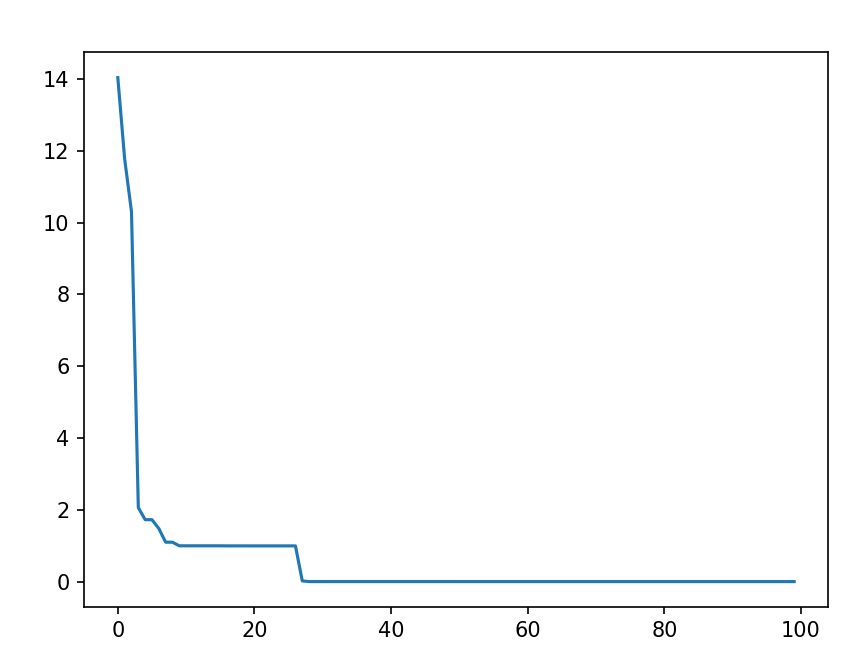


Рисунок 1.4.1 – Результат выполнения программы

# Заключение

В ходе выполнения данной работы выполнены поставленные задачи – изучен электромагнитный алгоритм, произведён его ручной расчёт для решения задачи поиска глобального минимума функции, а также разработаны программы на языке Python для нахождения глобального минимума функции Растригина от двух переменных.

В заключение можно отметить, что электромагнитный алгоритм является мощным инструментом для решения задач оптимизации (например, для нахождения глобального минимума функции от нескольких переменных), в которых стандартные методы недостаточно эффективны из-за наличия множества локальных минимумов. Алгоритм имеет меньшую зависимость от свободных параметров, чем пчелиный алгоритм, но большую, чем роевой алгоритм. В то же время, алгоритм показал быструю сходимость, однако имеет тенденцию сходится к локальным минимумам.

# Список информационных источников

1. Сорокин, А.Б. Введение в роевой интеллект: теория, расчеты и приложения [Электронный ресурс]: Учебно-методическое пособие / А.Б. Сорокин. — М.: Московский технологический университет (МИРЭА), 2019.
2. Карпенко, А.П. Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой: учебное пособие / А. П. Карпенко. – Москва: Издательство МГТУ им. Н. Э. Баумана, 2021. – 446 c.
3. Сорокин, А. Б. Безусловная оптимизация : учебно-методическое пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк ; — Москва : МИРЭА - Российский технологический университет, 2020.

# приложения

Приложение А — Реализация электромагнитного алгоритма в задаче оптимизации на языке Python.

### Приложение А

Реализация электромагнитного алгоритма в задаче оптимизации на языке Python.

Листинг А.1 – Реализация электромагнитного алгоритма

import numpy as np

import matplotlib.pyplot as plt

def rastrigin(x: np.ndarray):

    return np.sum(x\*\*2 - 10 \* np.cos(2 \* np.pi \* x) + 10)

class EMA:

    def \_\_init\_\_(self, n: int):

        self.n = n

        self.population\_size = 10 \* n

        self.max\_iter = 50 \* n

        self.local\_iter = 10

        self.scale = 0.005

        self.\_min = -5.12

        self.\_max = -self.\_min

    def calculate\_best(self):

        self.values = np.array([rastrigin(x) for x in self.x])

        self.best\_value = np.min(self.values)

        self.best\_x = self.x[np.where(abs(self.values - self.best\_value) < 1e-3)].flatten()

    def create\_population(self):

        self.x = np.vstack([self.\_min + np.random.uniform(0, 1, size=self.n) \* (self.\_max - self.\_min)

                           for \_ in range(self.population\_size)])

        self.calculate\_best()

    def local\_search(self):

        search\_field = self.scale \* (self.\_max - self.\_min)

        for k, particle in enumerate(self.x):

            cnt = 0

            while cnt < self.local\_iter:

                for i in range(self.n):

                    sign = np.random.randint(0, 2) \* 2 - 1

                    y = particle.copy()

                    velocity = np.random.uniform()

                    y[i] += sign \* velocity \* search\_field

                    if rastrigin(y) < rastrigin(particle):

                        self.x[k] = y.copy()

                        cnt = self.local\_iter

                        break

                    cnt += 1

        self.calculate\_best()

    def calculate\_force(self):

        self.q = np.exp(-self.n \* (self.values - self.best\_value) / (np.sum(self.values - self.best\_value)))

        self.force = np.zeros\_like(self.x)

        for i in range(self.population\_size):

            for j in range(self.population\_size):

                if i != j:

                    if self.values[j] < self.values[i]:

                        self.force[i] += ((self.x[j] - self.x[i]) /

Окончание Листинга А.1

np.linalg.norm(self.x[j] - self.x[i]) \*\* 2) \

                                         \* self.q[i] \* self.q[j]

                    else:

                        self.force[i] += ((self.x[i] - self.x[j]) / np.linalg.norm(self.x[j] - self.x[i]) \*\* 2) \

                                         \* self.q[i] \* self.q[j]

    def move\_particles(self):

        for i in range(self.population\_size):

            if abs(self.values[i] - self.best\_value) > 1e-3:

                alpha = np.random.uniform()

                velocity = np.ones\_like(self.x[i])

                normalized\_force = self.force[i] / np.linalg.norm(self.force[i])

                for j in range(self.n):

                    if self.force[i][j] > 0:

                        velocity[j] = self.\_max - self.x[i][j]

                    else:

                        velocity[j] = self.x[i][j] - self.\_min

                self.x[i] += alpha \* np.multiply(normalized\_force, velocity)

    def solve(self):

        self.create\_population()

        history = []

        for i in range(self.max\_iter):

            history.append(self.best\_value)

            print(f'Текущее лучшее значение: {round(self.best\_value, 4)} в точке {list(map(lambda x: round(x, 4), self.best\_x))}')

            print(f'Итерация: {i + 1}')

            self.local\_search()

            self.calculate\_best()

            self.calculate\_force()

            self.move\_particles()

        plt.plot(history)

        plt.show()

def main():

    ema = EMA(2)

    ema.solve()

if \_\_name\_\_ == '\_\_main\_\_':

    main()