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 №3**

**Дисциплина: Приложение нейросетевых алгоритмов**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Цель работы**

Получение знаний и практических навыков построения однослойных нейронных сетей.

**Задание**

Разработать алгоритм и программу решения задачи на языке Python без использования специализированных библиотек.

**Описание решения**

*Задача: нейронная сеть на основе радиально-симметричных функций как аппроксимирующая модель.*

Для решения данной задачи использовались 2 библиотеки: math и matplotlib.pyplot. Были написаны такие классы, как Neuron и Neural\_Network. Класс Neuron содержит поле center, а так же методы \_\_init\_\_() и get\_center() для инициализации и получения его поля из другого класса. Класс же Neural\_Network содержит только методы: gauss\_fun() – получение значения функции Гаусса; calculate\_characteristics\_matrix() – расчёт характеристической матрицы значений радиально-симметричных элементов; calculate\_matrix\_weight\_coefficients() – расчёт матрицы весовых коэффициентов; calculate\_total\_outputs() – расчёт списка итоговых выходов на основе матрицы весовых коэффициентов.

Алгоритм работы программы следующий:

- в качестве входных параметров у нас имеется список значений Х и список соответствующих им значений Y;

- берутся центры как 1, 3, 5, 7, 9 значения из списка X, радиус же можно взять одинаковый для всех радиальных элементов, равный 1.5;

- создаётся список объектов-нейронов и объект-нейронная сеть;

- заполняется матрица H на основе такой формулы:

- далее матрица H преобразуется в матрицу W:  


- на основе матрицы W рассчитываются итоговые значения, по которым мы строим график аппроксимирующей зависимости.

**Реализация программы на языке Python**

import math as mt

import matplotlib.pyplot as plt

*#Аналог функции transpose из numpy*

def matrix\_transposition(*matrix*):

    """

    Функция, которая транспонирует матрицу

    Args: matrix - исходная матрица

    Return: транспонированная матрица

    """

    row\_len = len(*matrix*)

    column\_len = len(*matrix*[0])

    result\_matrix = [[0 for i in range(row\_len)] for j in range(column\_len)]

    for i in range(row\_len):

        for j in range(column\_len):

            result\_matrix[j][i] = *matrix*[i][j]

    return result\_matrix

*#Аналог функции dot из numpy*

def matrix\_multiplication(*matrix\_1*, *matrix\_2*):

    """

    Функция, которая умножает 2 матрицы

    Args: matrix\_1 - первая матрица; matrix\_2 - вторая матрица

    Return: результирующая матрица

    """

    result\_matrix = [[0 for i in range(len(*matrix\_2*[0]))] for j in range(len(*matrix\_1*))]

    for i in range(len(*matrix\_1*)):

        for j in range(len(*matrix\_2*[0])):

            for k in range(len(*matrix\_2*)):

                result\_matrix[i][j] += *matrix\_1*[i][k] \* *matrix\_2*[k][j]

    return result\_matrix

*#Аналог функции inv из numpy.linalg*

*#Самая хитрая функция из всех, пусть будет с комментариями*

def rising\_matrix\_to\_minusOne(*matrix*):

    """

    Функция, которая возводит матрицу в степень -1 (находит обратную матрицу методом Гаусса-Жордана)

    Args: matrix - матрица

    Return: матрица, возведённая в степень -1 (обратная)

    """

    n = len(*matrix*)

    augmented\_matrix = [row + [1 if i == j else 0 for j in range(n)] for i, row in enumerate(*matrix*)]

*#Приведение к ступенчатому виду*

    for i in range(n):

*#Нормализация текущей строки*

        pivot = augmented\_matrix[i][i]

        for j in range(n \* 2):

            augmented\_matrix[i][j] /= pivot

*#Обнуление остальных строк*

        for k in range(n):

            if k != i:

                factor = augmented\_matrix[k][i]

                for j in range(n \* 2):

                    augmented\_matrix[k][j] -= factor \* augmented\_matrix[i][j]

*#Извлечение обратной матрицы из расширенной*

    inverse = [row[n:] for row in augmented\_matrix]

    return inverse

class Neuron:

    center = 0

    def \_\_init\_\_(*self*, *center*):

        """

        Функция, которая инициализирует нейрон с центром

        Args: center - центр

        Return: отсутствует

        """

*self*.center = *center*

    def get\_center(*self*):

        """

        Функция-гетер центра нейрона

        Args: отсутствуют

        Return: центр нейрона

        """

        return *self*.center

class Neural\_Network:

    def gauss\_fun(*self*, *x*, *center*, *radius*):

        """

        Функция Гаусса (как бы многомерный случай, но не совсем)

        Args: x - входное значение; center - центр соответствующего нейрона; radius - радиус

        Return: значение функции

        """

*#В многомерном случае в norm\_of\_vector корень из суммы квадратов по разным центрам*

        norm\_of\_vector = mt.pow(*x* - *center*, 2)

        a = 1 / (2 \* mt.pow(*radius*, 2))

        return mt.exp(-a \* norm\_of\_vector)

    def calculate\_characteristics\_matrix(*self*, *neurons\_quantity*, *entrances\_x*, *neurons*, *radius*):

        """

        Функция, которая считает характеристическую матрицу значений радиально-симметричных элементов

        Args: neurons\_quantity - кол-во нейронов; entrances\_x - входные значения; neurons - список нейронов; radius - радиус

        Return: характеристическая матрица

        """

        h = [[0 for j in range(*neurons\_quantity*)] for i in range(len(*entrances\_x*))]

        for i in range(len(*entrances\_x*)):

            for j in range(*neurons\_quantity*):

                h[i][j] += *self*.gauss\_fun(*entrances\_x*[i][0], *neurons*[j].get\_center(), *radius*)

        return h

    def calculate\_matrix\_weight\_coefficients(*self*, *matrix\_h*, *matrix\_y*):

        """

        Функция, которая считает матрицу весовых коэффициентов

        Args: matrix\_h - характеристическая матрица; matrix\_y - список правильных ответов

        Return: матрица весовых коэффициентов

        """

*#А это наглядный минус длинных названий функций)*

*#return matrix\_multiplication(matrix\_multiplication(rising\_matrix\_to\_minusOne(matrix\_multiplication(matrix\_transposition(matrix\_h), matrix\_h)), matrix\_transposition(matrix\_h)), matrix\_y)*

        h\_T\_h = matrix\_multiplication(matrix\_transposition(*matrix\_h*), *matrix\_h*)

        h\_minusOne = rising\_matrix\_to\_minusOne(h\_T\_h)

        h\_T\_y = matrix\_multiplication(matrix\_transposition(*matrix\_h*), *matrix\_y*)

        result = matrix\_multiplication(h\_minusOne, h\_T\_y)

        return result

    def calculate\_total\_outputs(*self*, *neurons*, *matrix\_w*, *entrances\_x*, *neurons\_quantity*, *radius*):

        """

        Функция, которая считает итоговые выходы на основе матрицы весовых коэффициентов

        Args: neurons - список нейронов; matrix\_w - матрица весовых коэффициентов; entrances\_x - входные значения; neurons\_quantity - кол-во нейронов; radius - радиус

        Return: список значений

        """

        total\_list = []

        for i in range(len(*entrances\_x*)):

            tmp = 0

            for j in range(*neurons\_quantity*):

                tmp += *self*.gauss\_fun(*entrances\_x*[i][0], *neurons*[j].get\_center(), *radius*) \* *matrix\_w*[j][0]

            total\_list.append(tmp)

        return total\_list

*#return [sum([self.gauss\_fun(entrances\_x[j], neurons[i].get\_center(), radius) \* matrix\_w[i][0] for i in range(neurons\_quantity)] for j in range(len(entrances\_x)))]*

neurons\_quantity = 5

entrances\_x = [[-2.0], [-1.5], [-1.0], [-0.5], [0.0], [0.5], [1.0], [1.5], [2.0]]

outputs\_y = [[-0.48], [-0.78], [-0.83], [-0.67], [-0.2], [0.7], [1.48], [1.17], [0.2]]

*#Взяли центры как 1, 3, 5, 7, 9 входной параметр*

centers = [-2.0, -1.0, 0.0, 1.0, 2.0]

*#Сказано, что r можно просто взять = 1.5 для всех нейронов*

radius = 1.5

neurons = [Neuron(centers[i]) for i in range(len(centers))]

neural\_network = Neural\_Network()

print()

print("Характеристическая матрица:")

h = neural\_network.calculate\_characteristics\_matrix(neurons\_quantity, entrances\_x, neurons, radius)

for i in range(len(h)):

    s = ""

    for j in range(len(h[0])):

        s += str(h[i][j]) + " "

    print(s)

print()

print("Матрица (вектор) весовых коэффициентов:")

w = neural\_network.calculate\_matrix\_weight\_coefficients(h, outputs\_y)

for i in range(len(w)):

    s = ""

    for j in range(len(w[0])):

        s += str(w[i][j]) + " "

    print(s)

print()

print("Вектор полученных значений:")

total\_outputs = neural\_network.calculate\_total\_outputs(neurons, w, entrances\_x, neurons\_quantity, radius)

for i in range(len(total\_outputs)):

    print(total\_outputs[i])

n = len(entrances\_x)

s = 0

for i in range(n):

    s += mt.fabs(1 - (outputs\_y[i][0] / total\_outputs[i]))

print(f"Средняя относительная ошибка аппроксимации: {s / n \* 100}%")

plt.scatter(entrances\_x, outputs\_y, *c*="blue", *label*="Исходные точки")

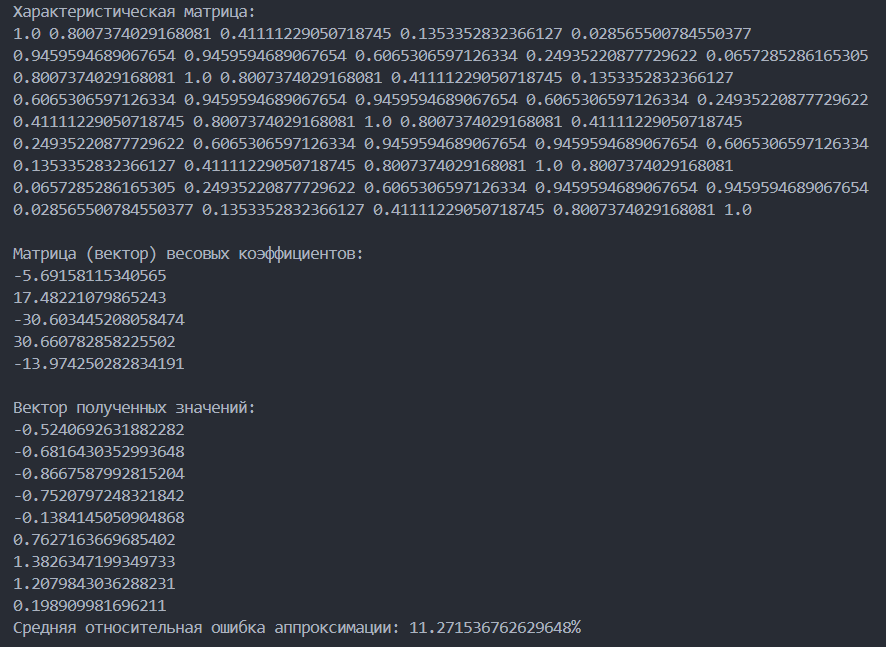
plt.plot(entrances\_x, total\_outputs, *label*="Полученная аппроксимирующая зависимость")

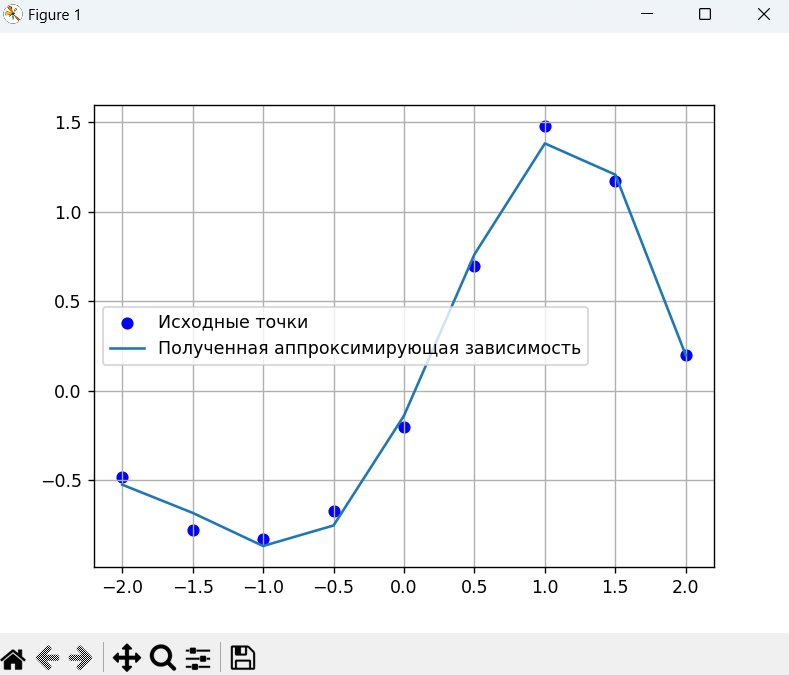
plt.legend()

plt.grid(True)

plt.show()

**Примеры запусков и выводы программы**





Небольшое замечание

Стоит сказать, что данная программа не то чтобы являлась нейросетью. От нейросети здесь нет ни весов, ни алгоритма обучения. Здесь происходят математические преобразования, которые никак не меняются от запуска к запуску. Ну либо это можно назвать «статичной нейросетью».