



Министерство науки и высшего образования Российской  
Федерации  
Федеральное государственное бюджетное образовательное  
учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы  
управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные  
технологии»

## Лабораторная работа № 2

**Тема** Построение и программная реализация алгоритма  
многомерной интерполяции табличных функций.

**Студент** Андреев А.А.

**Группа** ИУ7-44Б

**Оценка (баллы)** \_\_\_\_\_

**Преподаватель** \_\_\_\_\_

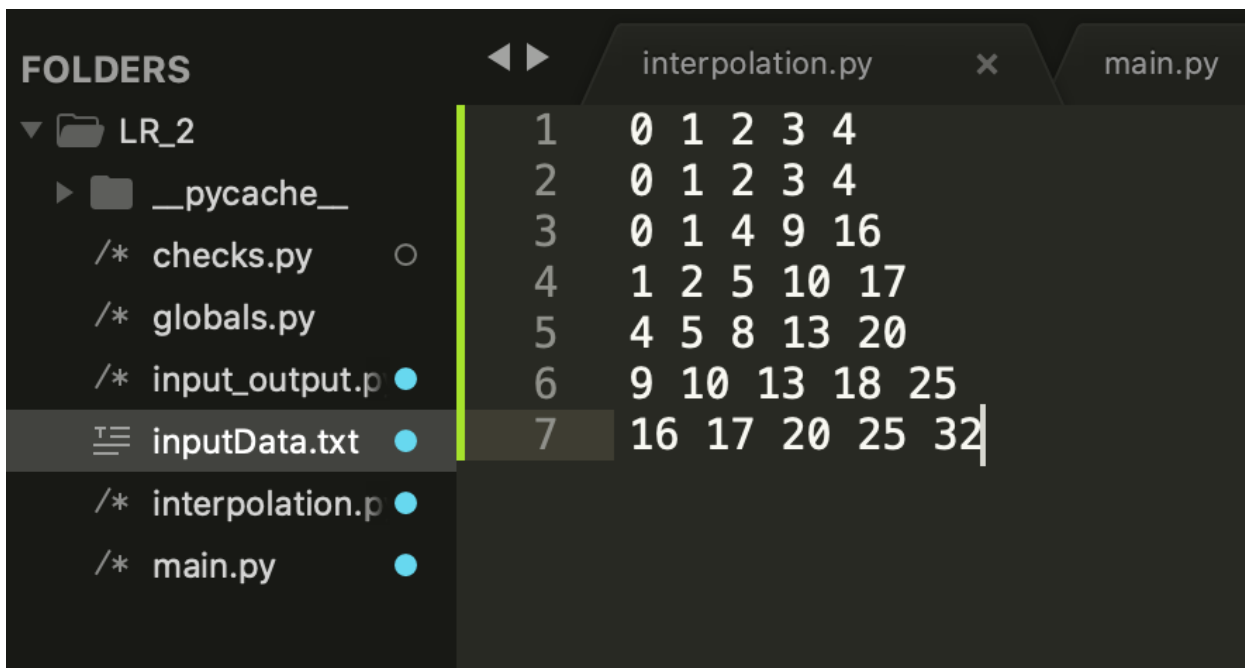
Москва.  
2021 г.

# Описание работы

**Тема:** Построение и программная реализация алгоритма многомерной интерполяции табличных функций.

**Цель работы:** Получение навыков построения алгоритма интерполяции таблично заданных функций двух переменных.

**Входные данные:** таблица точек, степень аппроксимирующих полиномов -  $p_x$  и  $p_y$ , значение аргументов  $x$ ,  $y$ , для которого выполняется интерполяция.



```
FOLDERS
└─ LR_2
   └─ __pycache__
      ├── /* checks.py
      ├── /* globals.py
      ├── /* input_output.p
      └─ inputData.txt
      ├── /* interpolation.p
      └─ /* main.py

interpolation.py
main.py

1  0  1  2  3  4
2  0  1  2  3  4
3  0  1  4  9 16
4  1  2  5 10 17
5  4  5  8 13 20
6  9 10 13 18 25
7 16 17 20 25 32
```

**Выходные данные:** Результат интерполяции  $z(x,y)$  при степенях полиномов 1,2,3 для  $x=1.5$ ,  $y=1.5$ .

## Алгоритм выполнения

### *Линейная интерполяция.*

Для нахождения полинома Ньютона первым делом выполняется сортировка входной таблицы по возрастанию аргументов. Данное действие необходимо для правильного выбора узлов, где  $X$  является центром конфигурации. Количество узлов равно степени полинома + 1. Для нахождения полинома  $n$ -ой степени необходимо найти разделенные разности.

Формулы вычисления разделенных разностей и интерполяционный многочлен Ньютона:

$$f(x_0; x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

$$f(x_0; x_1; x_2) = \frac{f(x_1; x_2) - f(x_0; x_1)}{x_2 - x_0} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0},$$

$$f(x_0; x_1; \dots; x_{n-1}; x_n) = \frac{f(x_1; \dots; x_{n-1}; x_n) - f(x_0; x_1; \dots; x_{n-1})}{x_n - x_0}.$$

$$L_n(x) = f(x_0) + f(x_0; x_1) \cdot (x - x_0) + f(x_0; x_1; x_2) \cdot (x - x_0) \cdot (x - x_1) + \dots + f(x_0; \dots; x_n) \cdot \prod_{k=0}^{n-1} (x - x_k)$$

После вычисления разделенных разностей, подставив аргумент в интерполяционный многочлен Ньютона можно найти значение функции.

В программе предусмотрено вычисление точного значения и относительной ошибки вычисления.

## ***Нахождение корня используя методы половинного деления и обратной интерполяции.***

Метод половинного деления позволяет исключать в точности половину интервала на каждой итерации. При использовании метода считается, что функция непрерывна и имеет на концах интервала разный знак. После вычисления значения функции в середине интервала одна часть интервала отбрасывается так, чтобы функция имела разный знак на концах оставшейся части. Итерации метода прекращаются если интервал становится достаточно мал (используется относительная погрешность)

Метод обратной интерполяции предусматривает перестановку столбцов аргументов и значений функции для нахождения методом линейной интерполяции значения аргумента при значении функции равном нулю.

# Код программы

Программа состоит из 4 файлов.

*main.py*

```
# Интерполяция таблично заданных функций полиномом Ньютона.
# Андреев Александр Алексеевич ИУ7-44Б.

import input_output, globals, checks, interpolation

# Основная функция программы
def main():
    if (input_output.getFileInputTableData(globals.resFileInputTableData, 5, 5, "inputData.txt") ==
        globals.ERROR_STATUS):
        return checks.outEndWorkingStatus(globals.ERROR_STATUS);

    if (input_output.getLevelOfAproximation(globals.nx, globals.ny) == globals.ERROR_STATUS):
        return checks.outEndWorkingStatus(globals.ERROR_STATUS);

    if (input_output.getValuesOfArguements(globals.x, globals.y) == globals.ERROR_STATUS):
        return checks.outEndWorkingStatus(globals.ERROR_STATUS);

    if (interpolation.getTrasmittedTable(globals.dataCompRow, globals.resFileInputTableData) ==
        globals.ERROR_STATUS):
        return checks.outEndWorkingStatus(globals.ERROR_STATUS);

    if (interpolation.getComparatorStatus(globals.dataCompRow, globals.resFileInputTableData,
        globals.x, globals.y, globals.nx, globals.ny, globals.functionValue, globals.newtonInterpolationValue) ==
        globals.ERROR_STATUS):
        return checks.outEndWorkingStatus(globals.ERROR_STATUS);

    input_output.outlineInterpolationCountingResult(globals.functionValue,
        globals.newtonInterpolationValue);

    return checks.outEndWorkingStatus(globals.SUCCESS_STATUS);

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

*input\_output.py*

```
import globals

# Чтение таблицы из файла
def getFileInputTableData(resFileInputTableData, numberOfRows, numberOfStrings, fileName):
    try:
        tempListOfStrings = str(open(fileName, "r").read()).split("\n");
        tempXValues = list(map(int, tempListOfStrings[0].split()));
        tempYValues = list(map(int, tempListOfStrings[1].split()));
        tempZValuesStrings = tempListOfStrings[2:];
        tempZValues = [];
        for tempString in tempZValuesStrings: tempZValues.append(list(map(int,
tempString.split()))))
        for tempX in range(len(tempXValues)):
            for tempY in range(len(tempYValues)):
                resFileInputTableData.append(globals.Point(int(tempXValues[tempX]),
int(tempYValues[tempY]), int(tempZValues[tempX][tempY])));
        return globals.SUCCESS_STATUS;
```

```

        except:
            return globals.ERROR_STATUS;

# Получение степени аппроксимации
def getLevelOfAproximation(nx, ny):
    try:
        globals.nx, globals.ny = list(map(float, input("Введите nx, ny: ").split()));
        return globals.SUCCESS_STATUS;
    except:
        return globals.ERROR_STATUS;

# Получение степени аппроксимации
def getValuesOfArguements(x, y):
    try:
        globals.x, globals.y = list(map(float, input("Введите x, y: ").split()));
        return globals.SUCCESS_STATUS;
    except:
        return globals.ERROR_STATUS;

# Вывод на экран результата
def outlineInterpolationCountingResult(functionValue, newtonInterpolationValue):
    print(str("\n\nРезультат функции: {} \nРезультат по интерполяции: {}".format(functionValue,
newtonInterpolationValue)));

```

### *globals.py*

```

# Статусы ошибок
ERROR_STATUS = 1
SUCCESS_STATUS = 0

# Степени аппроксимации
nx = 0;
ny = 0;

# Считанные значения x, y
x = 0;
y = 0;

# Данные файла
resFileInputTableData = list();

# Удобное для сдвига курсоров представление данных
dataCompRow = [[], [], []];

# Количество столбцов
numberOfCols = 5;

# Результаты вычисления
functionValue = 0
newtonInterpolationValue = 0

class Point():
    def __init__(self, xValue, yValue, zValue):
        self.xValue = xValue
        self.yValue = yValue
        self.zValue = zValue

```

```
import globals, input_output
import numpy as np
from math import *

# Получение значения обтекающей функции
def getFunction(x, y):
    return x ** 2 + y ** 2;

# Поиск ближайшего числа
def getNearNumbers(tempInputList, x):
    leftCur, rightCur = 0, len(tempInputList) - 1;
    while leftCur < rightCur:
        if tempInputList[int((leftCur + rightCur) / 2)] >= x: rightCur = int((leftCur + rightCur) / 2)
        else: leftCur = int((leftCur + rightCur) / 2) + 1
    return rightCur

# Рекурсивна обработка до единичного
def getCongestionValue(xValues, yValues):
    numberOfColls = len(xValues)
    while numberOfColls != 1:
        return (getCongestionValue(xValues[:-1], yValues[:-1]) - getCongestionValue(xValues[1:],
yValues[1:])) / (xValues[0] - xValues[numberOfColls - 1]);
    return yValues[0];

# Получение значения интерполяции Ньютона
def getNumberByNewtonInterpolationInSamples(tempInputListX, tempInputListZ, x):
    answerNumberByNewtonInterpolationInSamples = tempInputListZ[0]
    for rightCur in range(1, len(tempInputListX)):
        multipliedValue = 1
        for leftCur in range(rightCur): multipliedValue *= (x - tempInputListX[leftCur])
        answerNumberByNewtonInterpolationInSamples += (multipliedValue *
getCongestionValue(tempInputListX[:rightCur + 1], tempInputListZ[:rightCur + 1]))
    return answerNumberByNewtonInterpolationInSamples

# Преобразование таблицы в удобный вид
def getTrasmittedTable(dataCompRow, data):
    try:
        tempZline = [];
        for tempCur in range(len(data)):
            if (tempCur % globals.numberOfColls == 0):
dataCompRow[0].append(data[tempCur].xValue);
            if (tempCur < globals.numberOfColls):
dataCompRow[1].append(data[tempCur].yValue);
            if (tempCur % globals.numberOfColls == 0 and tempCur > 0):
                dataCompRow[2].append(tempZline);
                tempZline = [];
                tempZline.append(data[tempCur].zValue);
        return globals.SUCCESS_STATUS;
    except:
        return globals.ERROR_STATUS;

# Получение среднего для старта
def getMiddleCellStart(nearestList, nLevel):
    return nearestList - int(ceil((nLevel + 1) / 2));

# Получение среднего для конца
def getMiddleCellEnd(nearestList, nLevel):
    return nearestList + int(ceil((nLevel + 1) / 2));

# Установка всех промежутков для каждого этапа по X, Y, Z
```

```

def setAllSamples(allSamples, dataCompRow, dataSize, x, y, nx, ny):
    i_y = getNearNumbers(dataCompRow[1], y)

    if i_y - (ny + 1) / 2 < 0:
        allSamples[1], allSamples[2] = dataCompRow[1][:int(getMiddleCellEnd(i_y, ny) + 1)],
dataCompRow[2][:int(getMiddleCellEnd(i_y, ny) + 1)]
    elif dataSize < i_y + (ny + 1) / 2:
        allSamples[1], allSamples[2] = dataCompRow[1][getMiddleCellStart(i_y, ny):],
dataCompRow[2][getMiddleCellStart(i_y, ny):]
    else:
        if ny % 2 != 0:
            allSamples[1], allSamples[2] = dataCompRow[1][getMiddleCellStart(i_y, ny):
getMiddleCellEnd(i_y, ny)], dataCompRow[2][getMiddleCellStart(i_y, ny): getMiddleCellEnd(i_y, ny)]
        else:
            allSamples[1], allSamples[2] = dataCompRow[1][getMiddleCellStart(i_y, ny) -
1: getMiddleCellEnd(i_y, ny)], dataCompRow[2][getMiddleCellStart(i_y, ny) - 1: getMiddleCellEnd(i_y,
ny)]

# Сдвиг курсоров
def setAllCurs(allSamples, dataCompRow, dataSize, x, y, nx, ny):
    leftCur, rightCur = 0, 0;
    nearestXValueNumber = getNearNumbers(dataCompRow[0], x)
    if nearestXValueNumber - (nx + 1) / 2 < 0:
        allSamples[0] = dataCompRow[0][:int(getMiddleCellEnd(nearestXValueNumber, nx) +
1)]
        rightCur = getMiddleCellEnd(nearestXValueNumber, nx) + 1
    elif dataSize < nearestXValueNumber + (nx + 1) / 2:
        allSamples[0] = dataCompRow[0][getMiddleCellStart(nearestXValueNumber, nx):]
        leftCur, rightCur = getMiddleCellStart(nearestXValueNumber, nx), 6
    elif nx % 2 != 0:
        allSamples[0] = dataCompRow[0][getMiddleCellStart(nearestXValueNumber, nx):
getMiddleCellEnd(nearestXValueNumber, nx)]
        leftCur, rightCur = getMiddleCellStart(nearestXValueNumber, nx),
getMiddleCellEnd(nearestXValueNumber, nx)
    else:
        allSamples[0] = dataCompRow[0][getMiddleCellStart(nearestXValueNumber, nx) - 1:
getMiddleCellEnd(nearestXValueNumber, nx)]
        leftCur, rightCur = getMiddleCellStart(nearestXValueNumber, nx) - 1,
getMiddleCellEnd(nearestXValueNumber, nx)

    for tempCur in range(len(allSamples[2])):
        allSamples[2][tempCur] = allSamples[2][tempCur][int(leftCur):int(rightCur)]

# Заполнение массива значениями интерполяции
def getAnswerListOfNewtonInterpolations(allSamples, x):
    aswerListOfNewtonInterpolations = list();
    for tempCur in range(len(allSamples[1])):
        aswerListOfNewtonInterpolations.append(getNumberByNewtonInterpolationInSamples(allSamples[0],
allSamples[2][tempCur], x))
    return aswerListOfNewtonInterpolations;

# Основной компаратор вызова функций обработки
def getComparatorStatus(dataCompRow, data, x, y, nx, ny, functionValue, newtonInterpolationValue):
    try:
        allSamples = [], [], []
        setAllSamples(allSamples, dataCompRow, len(data), x, y, nx, ny);
        setAllCurs(allSamples, dataCompRow, len(data), x, y, nx, ny);
        globals.functionValue, globals.newtonInterpolationValue = getFunction(x, y),
getNumberByNewtonInterpolationInSamples(allSamples[1],
getAnswerListOfNewtonInterpolations(allSamples, x), y);
        return globals.SUCCESS_STATUS;

```

```
except:  
    return globals.ERROR_STATUS;
```

checks.py

```
import globals  
  
# Информирование о статусе завершения работы программы  
def outEndWorkingStatus(inputEndWorkingStatus):  
    if (inputEndWorkingStatus == globals.SUCCESS_STATUS):  
        print("Программа завершила работу корректно.");  
        return inputEndWorkingStatus;  
    print("Во время исполнения программы возникли ошибки.");  
    return inputEndWorkingStatus;
```

## Результат работы программы в таблице

Значения интерполяции при $x = 1.5, y = 1.5$			
$n_x / n_y$	1	2	3
1	5	4.875	4.75
2	4.875	4.75	4.625
3	4.75	4.625	4.5

Значения функции при $x = 1.5, y = 1.5$			
$n_x / n_y$	1	2	3
1	4.5		
2			
3			

## Контрольные вопросы:

1. Пусть производящая функция таблицы суть  $z(x,y)=x^2 + y^2$ . Область определения по  $x$  и  $y$  0-5 и 0-5. Шаги по переменным равны 1. Степени  $n_x = n_y = 1, x=y=1.5$ . Приведите по шагам те значения функции, которые получаются в ходе последовательных интерполяций. по строкам и столбцу.

Сначала я выставлю области значения по абсциссе и ординате:  $x, y \in [1, 2]$ , далее проведу интерполяцию по абсциссе:  $i = 1: 3.5, i = 2: 6.5$ , после по ординате, тогда получу значение равное пяти.



**2. Какова минимальная степень двумерного полинома, построенного на четырех узлах? На шести узлах?**

Количество узлов	Минимальная степень двумерного полинома
4	2
6	3

**3. Предложите алгоритм двумерной интерполяции при хаотичном расположении узлов, т.е. когда таблицы функции на регулярной сетке нет, и метод последовательной интерполяции не работает. Какие имеются ограничения на расположение узлов при разных степенях полинома?**

Когда у нас нет таблицы функций на регулярной сетке и метод последовательной интерполяции не работает, то можно найти коэффициенты, выбирая узлы, ближайшие к точке интерполяции, учитывая, что узлы полинома  $n$ -й степени должны лежать не на одной кривой, по 3-м узлам в окрестности точки интерполяции ( $z = a + b * x + c * y$ ) - это  $z_i$  для  $i$ -го узла от нуля до двух, то есть  $z_i = a + b * x_i + c * y_i$ , аналогично полином второй степени для  $i$ -узла от нуля до пяти.

**4. Пусть на каком-либо языке программирования написана функция, выполняющая интерполяцию по двум переменным. Опишите алгоритм использования этой функции для интерполяции по трем переменным.**

Пусть мы имеем  $f(x, y, z_i)$  и интерполяция по "i" - это  $\text{interpolation}(i, \dots)$ .

Тогда алгоритм: совершаем  $\text{interpolation}(z)$  один раз, которую получаем в ходе  $\text{interpolation}(x, y)$  в количестве  $n_z + 1$ .

**5. Можно ли при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней или даже разные методы одномерной интерполяции, например, полином Ньютона и сплайн?**

Да, при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней: Ньютона и сплайн возможно только в том случае, если точность вычислений, которая различная для каждого метода, будет учтена.

**6. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.**

Если взять треугольную конфигурацию узлов интерполяции, то число узлов -  $n$ , которое однозначно определяет многочлен  $m$  степени, который удобно записать в форме Ньютона, вводя разделенные разности функции двух переменных.

Обобщение одномерного варианта:

$$P_n(x, y) = \sum_{i=0}^n \sum_{j=0}^{n-1} z(x_0, \dots, x_i, y_0, \dots, y_j) \prod_{p=0}^{i-1} (x - x_p) \prod_{q=0}^{j-1} (y - y_q)$$