fМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ

КУРСОВАЯ РАБОТА

по курсу: Численные методы

на тему «Интерполирование функций двух переменных»

Выполнил: Группа: БПМ-18-2

Студент: Тишин Владислав Викторович

Проверил: Преподаватель: Рубчинский Александр Анатольевич

Москва, 2020 г.

Содержание

[Аннотация 3](#_Toc60257993)

[Введение 4](#_Toc60257994)

[Постановка математической задачи 4](#_Toc60257995)

[Изложение методов 5](#_Toc60257996)

[Проблема выбора узлов 5](#_Toc60257997)

[Билинейная интерполяция 6](#_Toc60257998)

[Пример 6](#_Toc60257999)

[Решение примера 6](#_Toc60258000)

[Интерполяционный многочлен Лагранжа 8](#_Toc60258001)

[Пример 8](#_Toc60258002)

[Решение примера 8](#_Toc60258003)

[Реализация методов 10](#_Toc60258004)

[Реализация билинейной интерполяции 10](#_Toc60258005)

[Вспомогательные классы 10](#_Toc60258006)

[Класс BilinearInterpolation 11](#_Toc60258007)

[Проверка работы 12](#_Toc60258008)

[Реализация интерполяционного многочлена Лагранжа 13](#_Toc60258009)

[Код функции 13](#_Toc60258010)

[Проверка работы 14](#_Toc60258011)

[Дополнения 15](#_Toc60258012)

[Выбор шага сетки 15](#_Toc60258013)

[Случай непрямоугольной сетки 15](#_Toc60258014)

[Список использованных источников 16](#_Toc60258015)

[Приложения 17](#_Toc60258016)

[Приложение 1 17](#_Toc60258017)

# Аннотация

В настоящем разделе рассматриваются метод билинейной интерполяции и интерполяционный многочлен Лагранжа для интерполяции функций двух переменных.

# Введение

**Интерполяция** - в вычислительной математике способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений.

## Постановка математической задачи

Рассмотрим систему несовпадающих точек из некоторой области . Пусть значения функции известны только в этих точках: .

* Точки называют узлами интерполяции, а их совокупность - интерполяционной сеткой.
* Тройки называют точками данных или базовыми точками.

Задача интерполяции состоит в поиске такой функции из заданного класса функций, что

* максимально приближает функцию в произвольной точке внутри интерполяционной сетки.

Функцию называют **интерполирующей функцией**.

# Изложение методов

## Проблема выбора узлов

Рассмотрим задачу интерполяции полиномами:

1. Заметим, что не любое число узлов интерполяции выгодно. Если для одной переменной степень многочлена была взаимно однозначно связана с числом узлов, то для двух переменных многочлен n-ой степени имеют узлов. Если число узлов не соответствует этой формуле, то часть коэффициентов при высших степенях должна задаваться принудительно (в частности, нулями): для выбора этих коэффициентов редко есть разумные основания.
2. Также не всякое расположение узлов допустимо: в одномерном случае узлы не должны были совпадать. Теперь же для интерполяции многочленом необходимо, чтобы узлы не лежали на прямой в плоскости . При интерполяции многочленом требуется, чтобы узлы не лежали на кривой -го порядка.

Поэтому для хорошей интерполяции сетка должна быть регулярно построенной, а не представлять собой совокупность беспорядочно расположенных точек. Следущие два примера используют прямоугольную сетку, образованную пересечением прямых

и

– значение функции в узле {}.

## Билинейная интерполяция

**Билинейной интерполяцией** называют расширение линейной интерполяции для функций двух переменных. Для начала реализуется линейная интерполяция по x на каждой прямой . Затем при каждом значении реализуется линейная интерполяция по с учетом значений функции, полученных на первом шаге.

Пусть тогда

Результат билинейной интерполяции не зависит от порядка шагов: можно сначала интерполировать вдоль оси абсцисс, а затем вдоль оси ординат, так и наоборот, результат будет одним и тем же.

### Пример

Пусть известны значения функции в следующих точках:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  |  |
| 0 | 5 | 10 |
|  | 20 | 40 | ? | 50 |
|  | 25 | ? | ? | ? |
|  | 30 | 30 | ? | 35 |

Необходимо найти значение функции в точке и промежуточные значения методом интерполирования.

### Решение примера

Промежуточные значения в точках вычисляются следующим образом:

Значение в точке вычислим с помощью билинейной интерполяции:

+

=

Итоговая таблица будет выглядеть следующим образом:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  |  |
| 0 | 5 | 10 |
|  | 20 | 40 | 45 | 50 |
|  | 25 | 35 | 38.75 | 42.5 |
|  | 30 | 30 | 32.5 | 35 |

## Интерполяционный многочлен Лагранжа

**Интерполяционный многочлен Лагранжа** - многочлен минимальной степени, принимающий данные значения в данном наборе точек.

Для двумерного случая многочлен будет выглядеть следующим образом:

при и

Базисные полиномы вычисляются по следующей формуле:

Отсюда следует, что , как линейная комбинация , может иметь степень не больше , и по определению

### Пример

Пусть известны значения функции в следующих точках:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  |  |  |
| 35 | 36 | 40 |
|  | 20 | 6 |  | 6.2 |
|  | 21 | 5.8 |  | 6 |
|  | 21.5 |  | ? |  |
|  | 22 | 5.5 |  | 5.8 |

Необходимо найти значение функции в точке

### Решение примера

Строим сначала интерполяционный многочлен Лагранжа по таблице:

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

Здесь . Очевидно, многочлен Лагранжа будет:

Далее, при строим многочлен Лагранжа используя таблицу:

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

Тогда,

Аналогично, при , получаем:

Найдём теперь После этого, строим таблицу по числам

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

По этой таблице строим многочлен

После этого, приближённо находим :

# Реализация методов

Мы реализовали два метода интерполирования функции двух переменных – билинейная интерполяция и интерполяционный многочлен Лагранжа. Реализация представляет из себя программный код, написанный на Python 3.9 в среде Google Colaboratory.

## Реализация билинейной интерполяции

### 

### Вспомогательные классы

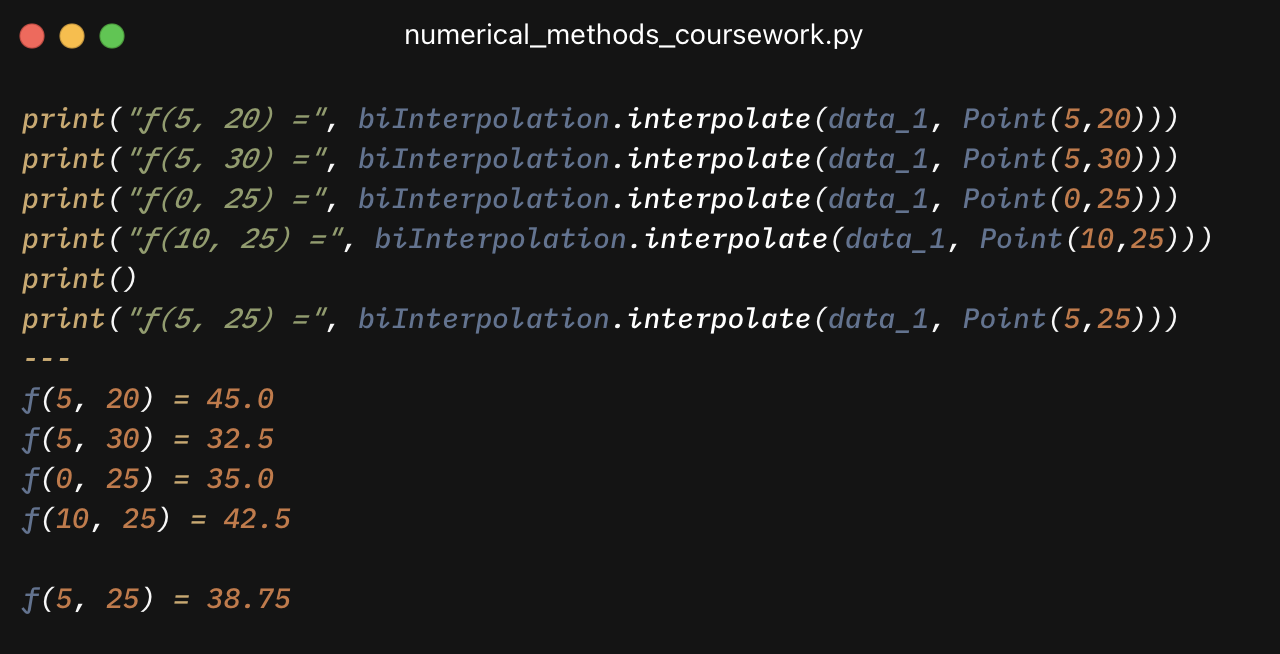


### Класс BilinearInterpolation



### Проверка работы

Воспроизведём пример из билинеарной интерполяции выше (см. стр. 7):

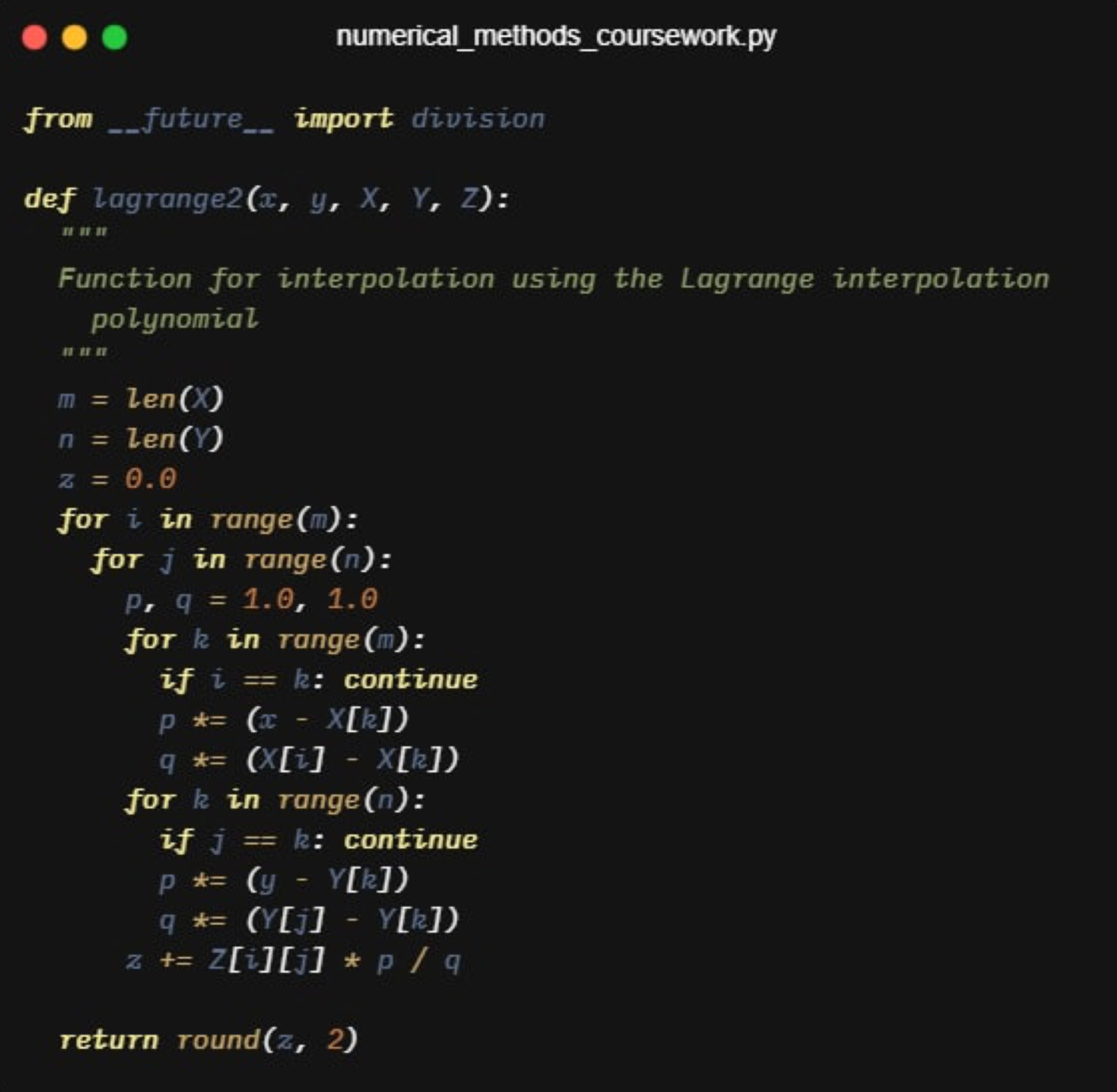


Как мы можем увидеть, наш алгоритм дал идентичный результат аналитическому методу.

Полный код прилагается в Приложение 1.

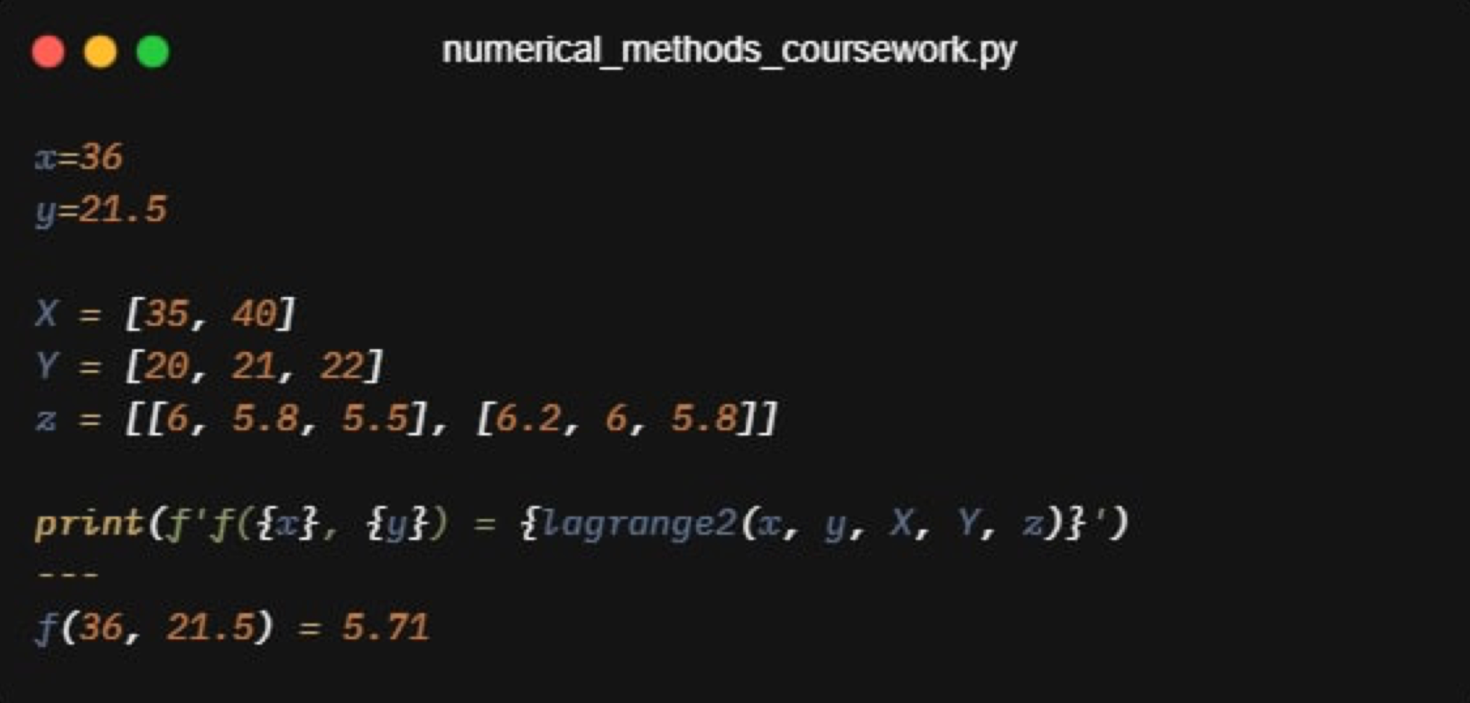
## Реализация интерполяционного многочлена Лагранжа

### Код функции



### Проверка работы

Воспроизведём пример из «Интерполяционный многочлен Лагранжа» выше (см. стр. 9):



Как мы можем увидеть, наш алгоритм дал идентичный результат аналитическому методу.

## Дополнения

### Выбор шага сетки

Если перед нами стоит задача интерполяции функции, как упрощение некой трудно вычислимой функции, т.е. мы сами можем выбирать узлы сетки и соответственно её шаг, то возникает вопрос - насколько нам дробить сетку? При делении сетки пополам можно рассматривать следующий параметр:

Также эти значения можно усреднить по всем узлам решётки, и при некотором его значении (например, ) прекратить деление.

### Случай непрямоугольной сетки

Такая ситуация возникает, когда базовыми точками, например, являются данные эксперимента, и нам необходимо проинтерполировать их на всё множество промежуточных значений. Тогда надо проводить интерполяцию по трем точкам: если — значение функции в вершинах , то вычислить приближенное значение функции внутри этого треугольника можно с помощью билинейной функции , находя коэффициенты из условий:

Где – координаты вершин Погрешность такой интерполяции для функции с непрерывными вторыми производными будет , где - длина наибольшей стороны треугольника

# Список использованных источников

1. *А.А.Самарский, А.В.Гулин.*  Численные методы М.: Наука, 1989.
2. *А.А.Самарский.*  Введение в численные методы М.: Наука, 1982.
3. *Н.Н.Калиткин.*  Численные методы М.: Наука, 1978.
4. *Н.С.Бахвалов, Н.П.Жидков, Г.М.Кобельков* Численные методы

# Приложения

## Приложение 1

Программный код BilinearInterpolation:

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_theme()

sns.set\_context("notebook")

from dataclasses import dataclass

@dataclass

class Point:

x: float = 0

y: float = 0

z: float = 0

@dataclass

class DataPoints:

points: np.array

@property

def x(self) -> np.array:

return np.unique(np.array([point.x for point in self.points]))

@property

def y(self) -> np.array:

return np.unique(np.array([point.y for point in self.points]))

@property

def z(self) -> np.array:

return np.unique(np.array([point.z for point in self.points]))

def get\_point(self, x: float, y: float) -> float:

for point in self.points:

if point.x == x and point.y == y:

return point.z

def construct\_points(x, y, z) -> np.array:

points = []

for idx\_x, x\_ in enumerate(x):

for idx\_y, y\_ in enumerate(y):

points.append(Point(x\_,y\_,z[idx\_x][idx\_y]))

return np.array(points)

def load\_data(file: str) -> DataPoints:

points = []

with open(file, "r") as f:

f.readline()

for line in f:

x, y, z = line.split(" ")

points.append(Point(float(x), float(y), float(z)))

result = DataPoints(np.array(points))

return result

class BilinearInterpolation:

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

pass

def \_\_find\_nearest\_points(self, data: DataPoints, point: Point):

nearest = []

for p, d in zip([point.x, point.y], [data.x, data.y]):

for idx, \_ in enumerate(d):

if d[idx] <= p <= d[idx+1]:

nearest.append((d[idx], d[idx+1]))

break

return nearest

def interpolate(self, data: DataPoints, point: Point):

is\_interpolation = min(data.x) <= point.x <= max(data.x) \

and min(data.y) <= point.y <= max(data.y)

assert is\_interpolation, "point is outside of the grid"

nearest\_x, nearest\_y = self.\_\_find\_nearest\_points(data, point)

z00 = data.get\_point(nearest\_x[0], nearest\_y[0])

z01 = data.get\_point(nearest\_x[0], nearest\_y[1])

z10 = data.get\_point(nearest\_x[1], nearest\_y[0])

z11 = data.get\_point(nearest\_x[1], nearest\_y[1])

alpha = np.round((point.x - nearest\_x[0]) / (nearest\_x[1] - nearest\_x[0]), 2)

beta = np.round((point.y - nearest\_y[0]) / (nearest\_y[1] - nearest\_y[0]), 2)

dx = z10 - z00

dy = z01 - z00

z = z00 + alpha \* dx + beta \* dy + alpha \* beta \* (z11 - dx - dy - z00)

return np.round(z, decimals=2)

x\_1 = [0, 10]

y\_1 = [20, 30]

z\_1 = [[40, 30], [50, 35]]

points\_1 = construct\_points(x\_1,y\_1,z\_1)

data\_1 = DataPoints(points=points\_1)

biInterpolation = BilinearInterpolation()

print("f(5, 20) =", biInterpolation.interpolate(data\_1, Point(5,20)))

print("f(5, 30) =", biInterpolation.interpolate(data\_1, Point(5,30)))

print("f(0, 25) =", biInterpolation.interpolate(data\_1, Point(0,25)))

print("f(10, 25) =", biInterpolation.interpolate(data\_1, Point(10,25)))

print()

print("f(5, 25) =", biInterpolation.interpolate(data\_1, Point(5,25)))