БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Отчёт по теме

«Аппроксимация и интерполяция функций»

Вариант 9

Выполнили: студенты гр. 853503

Климович А. А.

Галиева Э. Г. Ю.

Ивойлов О. А.

Проверил:

Протько М. И.

Минск 2020

**Содержание**

[Введение 3](#_Toc41506059)

[Конечная разность 3](#_Toc41506060)

[Разделённые разности 3](#_Toc41506061)

[Интерполяционный метод Лагранжа 5](#_Toc41506062)

[Интерполяционная формула Ньютона 5](#_Toc41506063)

[Интерполяция сплайнами 6](#_Toc41506064)

[Листинги программ и результаты 7](#_Toc41506065)

[Задание 1 7](#_Toc41506066)

[Код программы и результаты к заданию 1 7](#_Toc41506067)

[Результаты работы 11](#_Toc41506068)

[Метод наименьших квадратов 11](#_Toc41506069)

[Полиномиальный метод апроксимации 13](#_Toc41506070)

[Метод кубических сплайнов 15](#_Toc41506071)

[Задание 2 16](#_Toc41506072)

[Код программы и результаты к заданию 2 16](#_Toc41506073)

[Pезультаты работы 18](#_Toc41506074)

[Задание 3 19](#_Toc41506075)

[Код программы и результаты к заданию 3 19](#_Toc41506076)

[Результаты работы 21](#_Toc41506077)

[Вывод 23](#_Toc41506078)

# Введение

## Конечная разность

**.**

Конечная разность — [математический](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) термин, широко применяющийся в [методах вычисления](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) при [интерполировании](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F).

Рассмотрим интерполяционную задачу для функции :

, ..., ,

где , .

**Конечной разностью 1-го порядка** называют разность между двумя соседними значениями в узлах интерполяции, то есть

, .

**Конечной разностью 2-го порядка** называют разность между двумя соседними конечными разностями 1-го порядка, то есть

, .

**Конечной разностью порядка**  (для ) называют разность между двумя соседними конечными разностями порядка то есть

.

Если ввести оператор смещения такой, что , то оператор восходящей конечной разности и

,

который можно раскладывать по [биному Ньютона](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%BD%D0%BE%D0%BC_%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD%D0%B0). Данный способ представления заметно упрощает работу с конечными разностями высших порядков.

Конечные разности применяются в [интерполяционном методе Ньютона](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%84%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D1%8B_%D0%9D%D1%8C%D1%8E%D1%82%D0%BE%D0%BD%D0%B0). С конечными разностями связаны понятия [разделённых разностей](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%80%D0%B0%D0%B7%D0%BD%D0%BE%D1%81%D1%82%D1%8C) и [модуля непрерывности](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%D0%BD%D0%B5%D0%BF%D1%80%D0%B5%D1%80%D1%8B%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D0%B8).

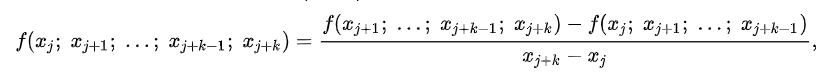
## Разделённые разности

**.**

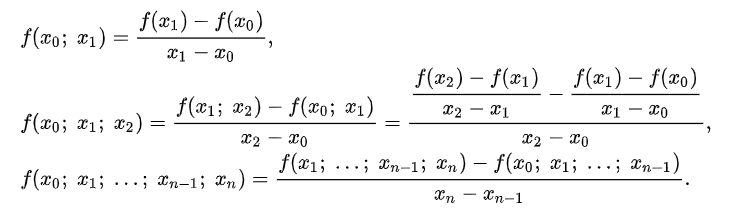
**Разделённая разность** — обобщение понятия [производной](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8) для дискретного набора точек.

Пусть функция задана на (связном) множестве и фиксированы попарно различныеточки , ..., .

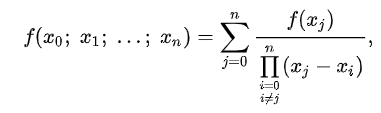
Тогда **разделённой разностью** **нулевого порядка** функции точке называют значение а **разделённую разность порядка** для системы точек определяют через разделённые разности порядка по формуле



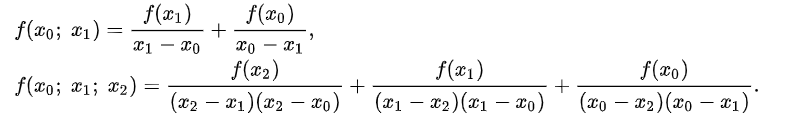
в частности,



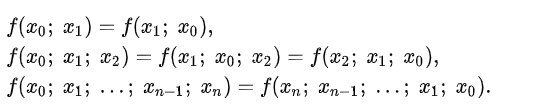
Для разделённой разности верна формула



в частности,



Разделённая разность является [симметрической функцией](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD) своих аргументов, то есть при любой их перестановке её значение не меняется, в частности,



При фиксированной системе точек разность является [линейным функционалом](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB), то есть для функций и скаляров и :



## Интерполяционный метод Лагранжа

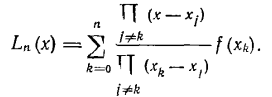
**.**

Интерполяционный многочлен Лагранжа — многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для *n + 1* пар чисел , где все различны, существует единственный многочлен *L(x)* степени не более *n*, для которого *L() =* .

В простейшем случае (*n = 1*) — это линейный многочлен, график которого — прямая, проходящая через две заданные точки.

Задача интерполирования многочленами состоит в том, чтобы построить многочлен:

значения которого в заданных точках *x* совпадают со значениями функции *f(x)* в этих точках.

Формула для нахождения многочлена:

где и — заданные точки (узлы).

## Интерполяционная формула Ньютона

**.**

Данная формула позволяет выразить интерполяционный многочлен через значение в одном из узлов и через разделенные разности функции , построенные по узлам . Она является разностным аналогом формулы Тейлора.

Формула разделённых разностей:

Интерполяционным многочленом Ньютона называется многочлен:

## Интерполяция сплайнами

**.**

**Сплайн** — [функция](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), область определения которой разбита на конечное число отрезков, на каждом из которых она совпадает с некоторым алгебраическим [многочленом](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D1%87%D0%BB%D0%B5%D0%BD) ([полиномом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC)). Максимальная из степеней использованных полиномов называется **степенью сплайна**. Разность между степенью сплайна и получившейся [гладкостью](https://ru.wikipedia.org/wiki/%D0%93%D0%BB%D0%B0%D0%B4%D0%BA%D0%B0%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F) называется **дефектом сплайна**. Например, непрерывная ломаная есть сплайн степени 1 и дефекта 1. В современном понимании сплайны — это решения многоточечных [краевых задач](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D0%B5%D0%B2%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) сеточными методами.

Другими словами сплайн — это кусочно заданная функция, то есть совокупность нескольких функций, каждая из которых задана на каком-то множестве значений аргумента, причём эти множества попарно непересекающиеся.

Сплайны имеют многочисленные применения как в математической теории, так и в [прикладной математике](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) (в частности, в разнообразных вычислительных программах). В частности, сплайны двух переменных интенсивно используются для задания поверхностей в различных системах [компьютерного моделирования](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Сплайны двух аргументов называют би-сплайнами (например, бикубический сплайн), которые являются двумерными сплайнами, моделирующими поверхности. Их часто путают с [B-сплайнами](https://ru.wikipedia.org/wiki/B-%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD) (базисными сплайнами), которые являются одномерными и в линейной комбинации составляют кривые — каркас для «натягивания» поверхностей. Также из базисных сплайнов возможно составить трёхмерную конструкцию для моделирования объёмных тел.

# Листинги программ и результаты

## Задание 1

**.**

Для заданных на отрезке функций

1. , , , , ;
2. , , , , ;
3. , , , , , ;
4. , , , , , ;

используйте три различных метода аппроксимации для оценки , , , , , . Сравните результаты по абсолютной ошибке.

## Код программы и результаты к заданию 1

**:**

import matplotlib.pyplot as plt  
from matplotlib import rcParams  
import math as m  
import sympy as sym  
import numpy as np  
import pandas as pd  
import warnings  
from scipy.interpolate import CubicSpline  
  
  
def GraphInterpretation(X: list, Y: list, f: list, names: list):  
 rborder = 1  
 rcParams['figure.subplot.right'] = rborder  
 rcParams['figure.subplot.hspace'] = 0.40  
 rcParams['figure.subplot.wspace'] = 0.40  
  
 fig = plt.figure()  
 fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 7))  
  
 for i, ax in enumerate(fig.axes):  
 minimum = min(X[i])  
 maximum = max(X[i])  
 koeff = (maximum - minimum) / 100  
 x = [minimum + koeff \* i for i in range(100)]  
 ax.plot(x, [f[i](j) for j in x], label='Аппроксимация', c='orange')  
 ax.scatter(X[i], Y[i], label='Значения функции', s=15, c='brown')  
 ax.title.set\_text(names[i])  
 ax.grid(True)  
 ax.set\_xlabel(u'X')  
 ax.set\_ylabel(u'Y')  
 ax.legend(loc='best', frameon=False) # легенда для области рисования ax  
 plt.show()

def CreateMatrix(X: list, Y: list, N: int, degree: int):  
 S = [[0.] \* (degree + 1) for i in range(degree + 1)]  
 S[0][0] = N  
 for i in range(1, 2 \* (degree) + 1):  
 value = sum(x \*\* i for x in X)  
 for j in range(i + 1):  
 if j < degree + 1 and i - j < degree + 1 and i - j >= 0:  
 S[i - j][j] = value  
 B = [0] \* (degree + 1)  
 for i in range(degree + 1):  
 B[i] = sum(Y[j] \* X[j] \*\* i for j in range(N))  
 # print('S:\n{}\n\nB:\n{}0'.format(S,B))  
 return np.array(S), np.array(B)  
  
  
def LeastSquareMethod(X: list, Y: list, N: int):  
 S, B = CreateMatrix(X, Y, N, 1)  
 A = np.linalg.inv(S).dot(B)  
 func = lambda x: float(A[0] + A[1] \* float(x))  
 return func  
  
  
def PolynomApproximation(X: list, Y: list, N: int, degree: int):  
 if degree <= 0 or degree > 10:  
 raise AttributeError('The degree of the polynomial must be within (1;10). Not {}.'.format(B))  
 S, B = CreateMatrix(X, Y, N, degree)  
 A = np.linalg.inv(S).dot(B)  
 func = lambda x: A[0] + sum(A[i] \* x \*\* i for i in range(1, degree + 1))  
 return func  
  
  
class Spline:  
 def \_\_init\_\_(self, a, b, c, d, x):  
 self.a = a  
 self.b = b  
 self.c = c  
 self.d = d  
 self.x = x  
  
  
def build\_spline(x, y, n):  
 splines = [Spline(0, 0, 0, 0, 0) for \_ in range(0, n)]  
 for i in range(0, n):  
 splines[i].x = x[i]  
 splines[i].a = y[i]  
 alpha = [0.0 for \_ in range(0, n - 1)]  
 beta = [0.0 for \_ in range(0, n - 1)]  
 splines[0].c = splines[n - 1].c = 0.0  
 for i in range(1, n - 1):  
 h = x[i] - x[i - 1]  
 h1 = x[i + 1] - x[i]  
 A = h  
 C = 2.0 \* (h + h1)  
 B = h1  
 F = 6.0 \* ((y[i + 1] - y[i]) / h1 - (y[i] - y[i - 1]) / h)  
 z = (A \* alpha[i - 1] + C)  
 alpha[i] = -B / z  
 beta[i] = (F - A \* beta[i - 1]) / z  
  
 for i in range(n - 2, 0, -1):

splines[i].c = alpha[i] \* splines[i + 1].c + beta[i]  
  
 for i in range(n - 1, 0, -1):  
 h = x[i] - x[i - 1]  
 splines[i].d = (splines[i].c - splines[i - 1].c) / h  
 splines[i].b = h \* (2.0 \* splines[i].c + splines[i - 1].c) / 6.0 + (y[i] - y[i - 1]) / h  
 func = lambda x: interpolate(splines, x)  
 return func  
  
  
def interpolate(splines, x):  
 if not splines:  
 return None  
 n = len(splines)  
 s = Spline(0, 0, 0, 0, 0)  
  
 if x <= splines[0].x:  
 s = splines[0]  
 elif x >= splines[n - 1].x:  
 s = splines[n - 1]  
 else:  
 i = 0  
 j = n - 1  
 while i + 1 < j:  
 k = i + (j - i) // 2  
 if x <= splines[k].x:  
 j = k  
 else:  
 i = k  
 s = splines[j]  
  
 dx = x - s.x  
 return s.a + (s.b + (s.c / 2.0 + s.d \* dx / 6.0) \* dx) \* dx  
  
  
def CountAndPlotApproximations(X, Y,test,func, degree):  
 name = ['e^(2x)\*cos(3x)', 'sin(ln(x))', 'ln(x)', 'cos(x) + sin(x)']  
 LeastSqare\_res = [LeastSquareMethod(X[0], Y[0], len(X[0])), LeastSquareMethod(X[1], Y[1], len(X[1])),  
 LeastSquareMethod(X[2], Y[2], len(X[2])), LeastSquareMethod(X[3], Y[3], len(X[3]))]  
 GraphInterpretation(X, Y, LeastSqare\_res, name)  
 print("None")   
 for i in range(0,4):  
 print("\nФункция:{0}".format(name[i]))  
 for j in range(0, len(test)):  
 print("S({0}) = {1}".format(test[j],LeastSqare\_res[i](test[j])))  
 print("delta({0}) = {1}".format(test[j], abs(LeastSqare\_res[i](test[j]) - func[i](test[j]))))  
 PolynomApproximation\_res = [PolynomApproximation(X[0], Y[0], len(X[0]), degree),  
 PolynomApproximation(X[1], Y[1], len(X[1]), degree),  
 PolynomApproximation(X[2], Y[2], len(X[2]), degree),  
 PolynomApproximation(X[3], Y[3], len(X[3]), degree)]  
 print("None")   
 for i in range(0, 4):  
 print("\nФункция:{0}".format(name[i])) for j in range(0, len(test)):  
 print("S({0}) = {1}".format(test[j], PolynomApproximation\_res[i](test[j])))  
 print("delta({0}) = {1}".format(test[j], abs(PolynomApproximation\_res[i](test[j]) - func[i](test[j]))))  
  
  
 GraphInterpretation(X, Y, PolynomApproximation\_res, name)  
  
 Square = [build\_spline(X[0], Y[0], len(X[0])),  
 build\_spline(X[1], Y[1], len(X[1])),  
 build\_spline(X[2], Y[2], len(X[2])),  
 build\_spline(X[3], Y[3], len(X[3]))]  
 print("Кубические сплайны")  
 for i in range(0, 4):  
 print("\nФункция:{0}".format(name[i]))  
 for j in range(0, len(test)):  
 print("S({0}) = {1}".format(test[j], Square[i](test[j])))  
 print("delta({0}) = {1}".format(test[j], abs(Square[i](test[j]) - func[i](test[j]))))  
  
 GraphInterpretation(X, Y, Square, name)  
  
  
f\_1 = lambda x: m.exp(2 \* x) \* m.cos(3 \* x)  
f\_2 = lambda x: m.sin(m.log(x))  
f\_3 = lambda x: m.log(x)  
f\_4 = lambda x: m.cos(x) + m.sin(x)  
test = [0.25,0.51,0.99,1.09,1.89,2.39]  
X\_values = [[0, 0.3, 0.6], [2.0, 2.4, 2.6], [1.0, 1.1, 1.3, 1.4], [0.0, 0.25, 0.5, 1.0]]  
Y\_values = [list(f\_1(x) for x in X\_values[0]), list(f\_2(x) for x in X\_values[1]),  
 list(f\_3(x) for x in X\_values[2]), list(f\_4(x) for x in X\_values[3])]  
CountAndPlotApproximations(X\_values, Y\_values, test,[f\_1,f\_2,f\_3,f\_4] ,2)

## Результаты работы

**:**

#### Метод наименьших квадратов

**:**

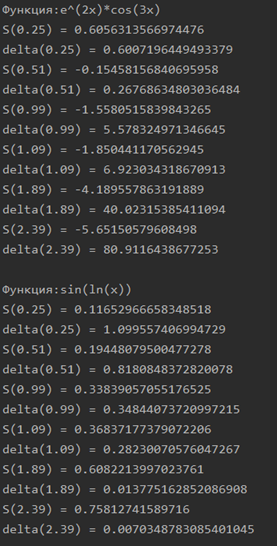


Рисунок 1. Результат работы для функций а и б.

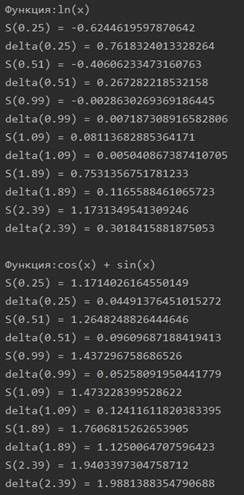


Рисунок 2. Результат работы для функций в и г.

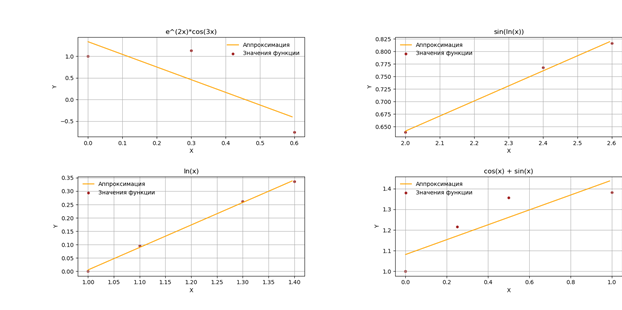


Рисунок 3. Графическое представление.

#### Полиномиальный метод апроксимации

:

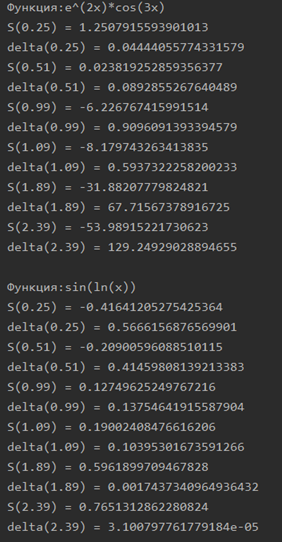


Рисунок 4. Результат работы для функций а и б.

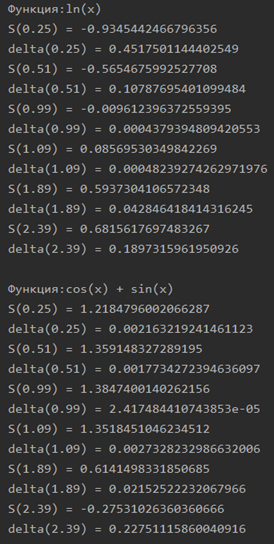


Рисунок 5. Результат работы для функций в и г.

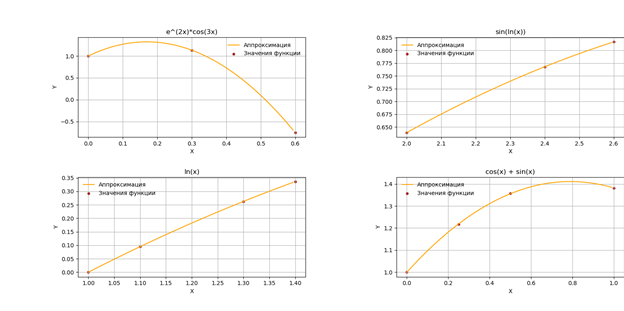


Рисунок 6. Графическое представление.

#### Метод кубических сплайнов

:

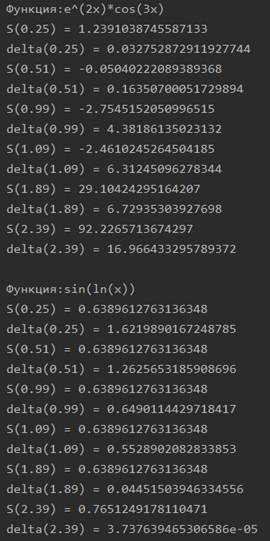


Рисунок 7. Результат работы для функций а и б.

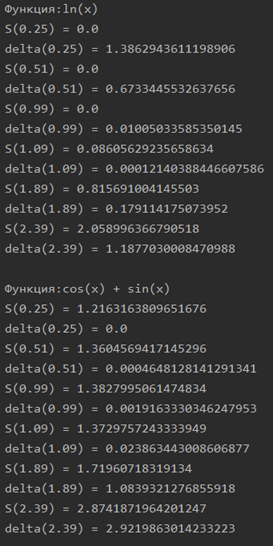


Рисунок 8. .Результат работы для функций в и г.

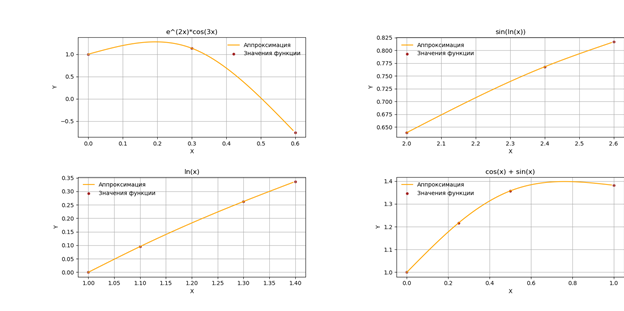


Рисунок 9. Графическое представление.

## **Задание 2**

**.**

Используйте исторические данные о дневном приросте количества заболевших в мире (<https://www.worldometers.info/coronavirus/worldwide-graphs/#daily-cases>) для оценки динамики заболеваемости методом наименьших квадратов и сплайнами.

## Код программы и результаты к заданию 2

**:**

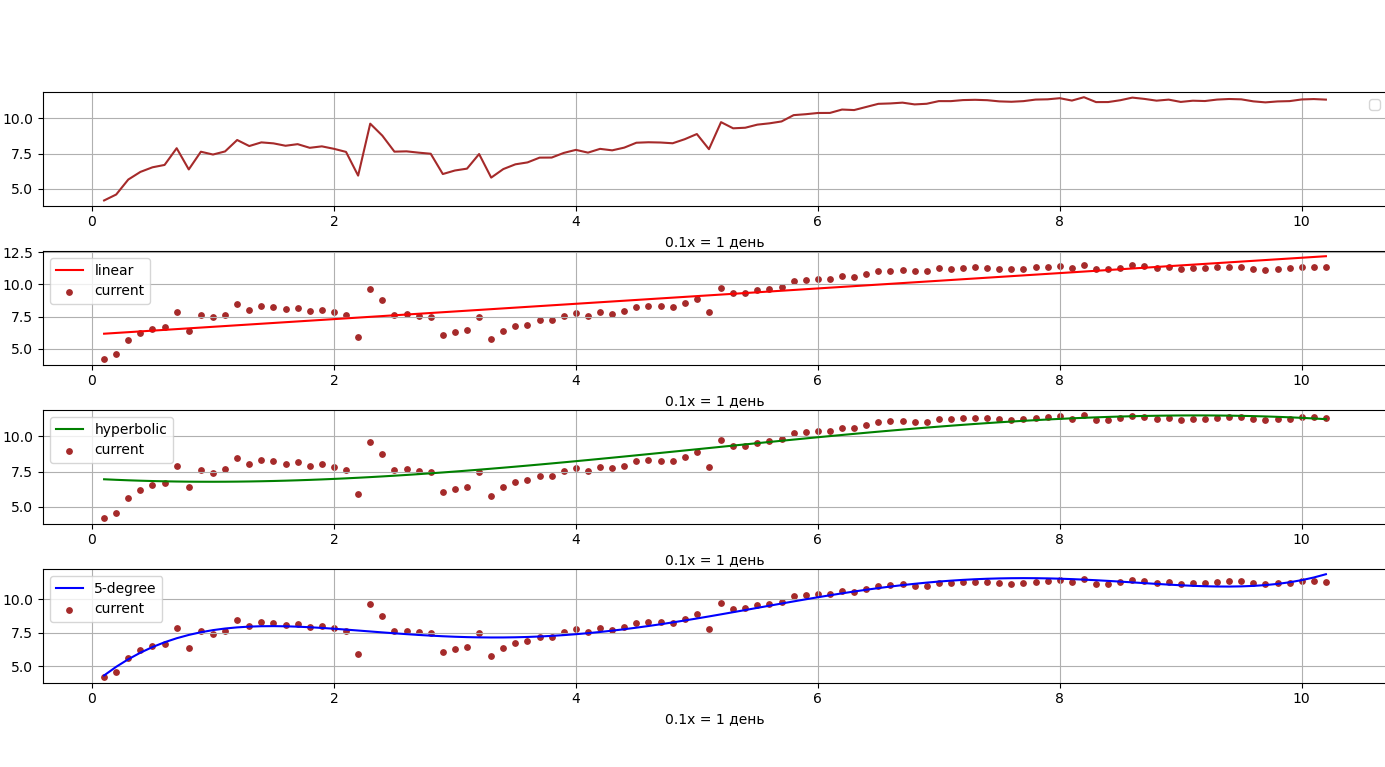
import matplotlib.pyplot as plt  
from matplotlib import rcParams  
import math as m  
import sympy as sym  
import numpy as np  
import pandas as pd  
import warnings  
from scipy.interpolate import CubicSpline  
  
  
def GetDataForAnalys(data: pd.core.frame.DataFrame):  
 daily\_sum\_list = [m.log(65)]  
 sum\_prev = sum(data['0122'])  
 headers = list(data)  
 for i in headers[2:]:  
 sum\_curr = sum(data[i])  
 daily\_sum\_list.append(m.log(sum\_curr - sum\_prev))  
 sum\_prev = sum\_curr  
 return daily\_sum\_list  
  
  
colors = {0: 'brown', 1: 'red', 2: 'green', 3: 'blue', 4: 'orange'}  
  
  
def PolynomApproximation(X: list, Y: list, N: int, degree: int):  
 if degree <= 0 or degree > 10:

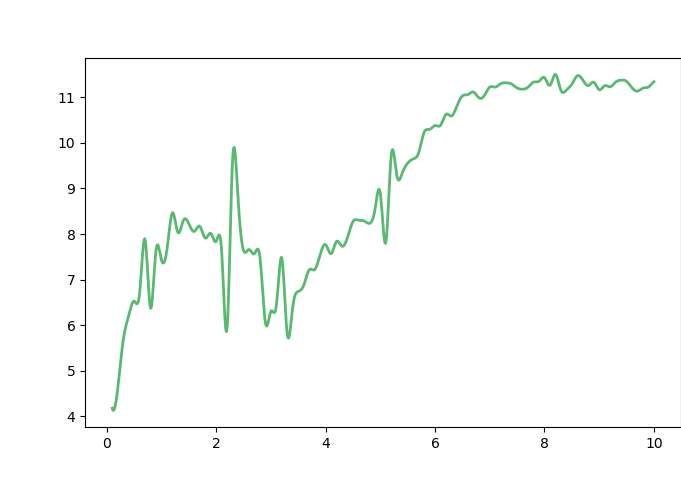
raise AttributeError('The degree of the polynomial must be within (1;10). Not {}.'.format(B))  
 S, B = CreateMatrix(X, Y, N, degree)  
 A = np.linalg.inv(S).dot(B)  
 func = lambda x: A[0] + sum(A[i] \* x \*\* i for i in range(1, degree + 1))  
 return func  
  
  
def CreateMatrix(X: list, Y: list, N: int, degree: int):  
 S = [[0.] \* (degree + 1) for i in range(degree + 1)]  
 S[0][0] = N  
 for i in range(1, 2 \* (degree) + 1):  
 value = sum(x \*\* i for x in X)  
 for j in range(i + 1):  
 if j < degree + 1 and i - j < degree + 1 and i - j >= 0:  
 S[i - j][j] = value  
 B = [0] \* (degree + 1)

for i in range(degree + 1):  
 B[i] = sum(Y[j] \* X[j] \*\* i for j in range(N))  
 # print('S:\n{}\n\nB:\n{}0'.format(S,B))  
 return np.array(S), np.array(B)  
  
  
def DisplayCoronaResults(X: list, Y: list, titles: list):  
 rborder = 1  
 rcParams['figure.subplot.right'] = rborder  
 rcParams['figure.subplot.hspace'] = 0.40  
 rcParams['figure.subplot.wspace'] = 0.40  
  
 fig, axes = plt.subplots(nrows=4, ncols=1, figsize=(15, 20))  
  
 for i, ax in enumerate(fig.axes):  
 if i == 0:  
 ax.plot(X, Y[0], c='brown')  
 else:  
 ax.scatter(X, Y[0], s=15, c=colors[0], label=titles[0])  
 ax.plot(X, Y[i], c=colors[i], label=titles[i])  
 ax.grid(True)  
 ax.set\_xlabel(u'0.1х = 1 день')  
 ax.set\_ylabel(u'logn(число заболевших)')  
 ax.legend(loc='best', frameon=True)  
 plt.show()  
 spline = CubicSpline(X, Y[0])  
 x = np.linspace(0.1, 10, 1000)  
 plt.plot(x, spline(x), color='#58b970', linewidth=2)  
 plt.show()  
  
  
data\_for\_analys = pd.read\_excel('D:/Countries-Confirmed.xlsx', header=0)  
daily\_sum\_list = GetDataForAnalys(data\_for\_analys)  
X = list(range(1, len(daily\_sum\_list) + 1))  
X = [(x \* 1.0) / 10 for x in X]  
  
Y = [daily\_sum\_list]  
for i in range(1, 8, 2):  
 f = PolynomApproximation(X, daily\_sum\_list, len(daily\_sum\_list), i)  
 Y.append([f(x) for x in X])  
  
DisplayCoronaResults(X, Y, ['current', 'linear', 'hyperbolic', '5-degree', '7-degree'])

## Pезультаты работы

**:**





## Задание 3

**.**

Аппроксимируйте форму буквы “N”. Используйте кривые Безьера и следующие данные:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 0 | 3 | 6 | 3.3 | 6.5 |  |  |
| 1 | 2 | 2 | 2.8 | 3.0 | 2.5 | 2.5 |
| 2 | 6 | 6 | 5.8 | 5.0 | 5.0 | 5.8 |
| 3 | 5 | 2 | 5.5 | 2.2 | 4.5 | 2.5 |
| 4 | 6.5 | 3 |  |  | 6.4 | 2.8 |

По аналогичной методологии аппроксимируйте первую букву своего имени.

## Код программы и результаты к заданию 3

**:**

import matplotlib.path as mpath  
import matplotlib.patches as mpatches  
import matplotlib.pyplot as plt  
import numpy as np  
import math  
  
# N  
  
x = [3, 3.3, 2.5, 2]  
y = [6, 6.5, 2.5, 2]  
  
binom = lambda n, i: math.factorial(n) / (math.factorial(i) \* math.factorial(n - i))  
Bt = lambda n, p, t: sum([binom(n, i) \* ((1 - t) \*\* (n - i)) \* (t \*\* i) \* p[i] for i in range(0, n + 1)])  
  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
x = [2, 2.8, 5, 6]  
y = [2, 3, 5.8, 6]  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]

plt.plot(bx, by)  
plt.plot(x, y, 'o')  
x = [6, 5.8, 4.5, 5]  
y = [6, 5, 2.5, 2]  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
x = [5, 5.5, 6.4, 6.5]  
y = [2, 2.2, 2.8, 3]  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
plt.show()  
  
# A  
x = [2, 2.5, 3.5, 5]  
y = [2, 4, 4, 7]  
  
binom = lambda n, i: math.factorial(n) / (math.factorial(i) \* math.factorial(n - i))  
Bt = lambda n, p, t: sum([binom(n, i) \* ((1 - t) \*\* (n - i)) \* (t \*\* i) \* p[i] for i in range(0, n + 1)])  
  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
x = [5, 6.5, 7.5, 8]  
y = [7, 4, 4, 2]  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
x = [3, 5, 5, 7]  
y = [4, 4.5, 3.5, 4]  
n = len(x) - 1  
bx = [Bt(n, x, t) for t in np.arange(0, 1., 0.01)]  
by = [Bt(n, y, t) for t in np.arange(0, 1., 0.01)]  
  
plt.plot(bx, by)  
plt.plot(x, y, 'o')  
plt.show()

## Результаты работы

**:**

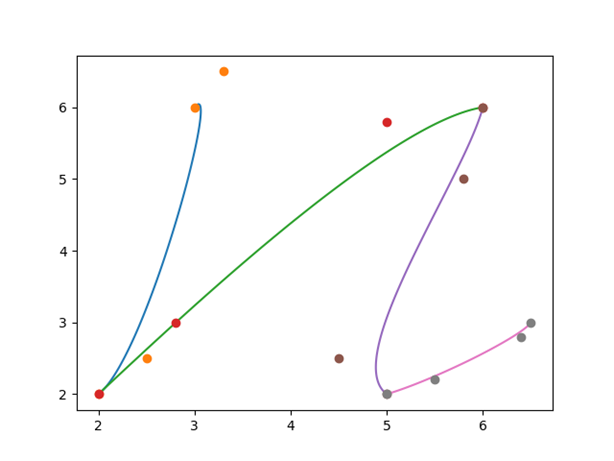


Рисунок 10. Аппроксимация буквы «И».

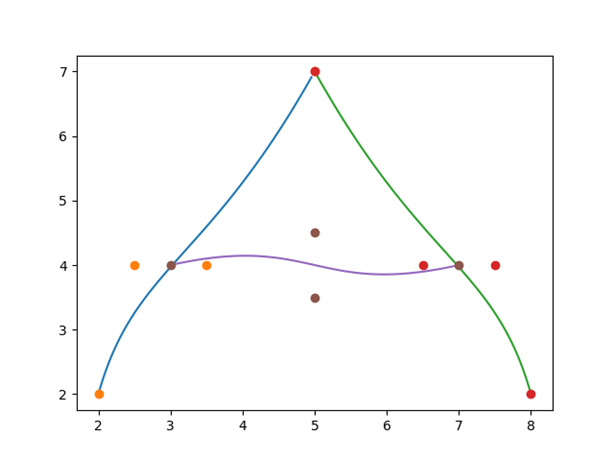


Рисунок 11. Аппроксимация буквы «А».

# Вывод

В процессе выполнения данной лабораторной работы нами были изучены методы интерполяции и аппроксимации. Было обнаружено, что метод кубический сплайнов дает наиболее точные результаты при аппроксимации функций.