**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе № 3**

**по дисциплине «Вычислительная математика»**

**Тема: Интерполяционная формула Ньютона**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Лисс А.Р. |

Санкт-Петербург

2022

**Цель работы:**

Рассмотреть метод интерполяции с помощью многочленов и использованием формулы Ньютона для равноотстоящих узлов.

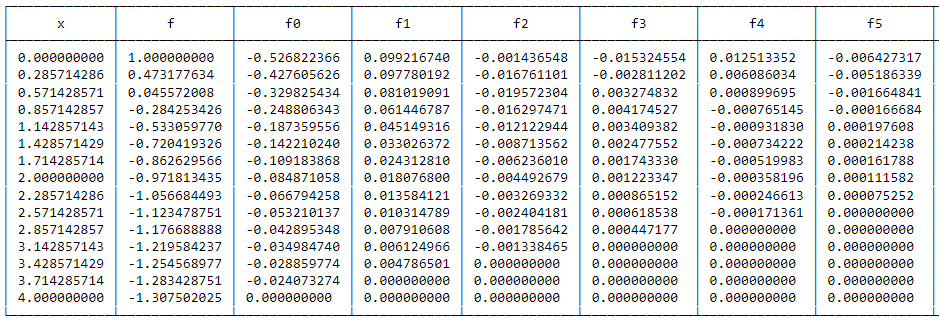
**Общие сведения:**

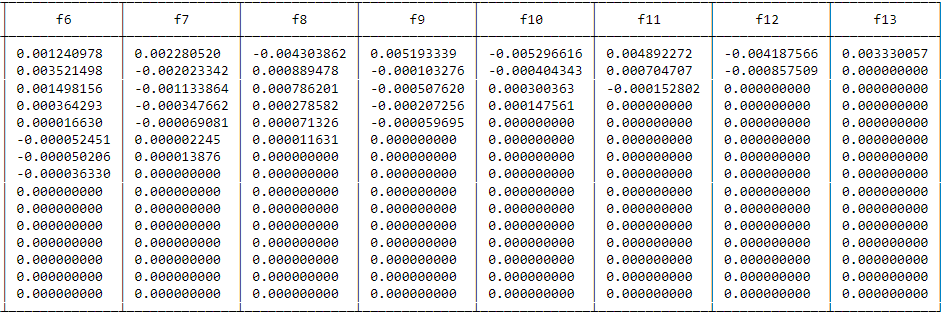
Алгоритм подразделяется на следующие этапы:

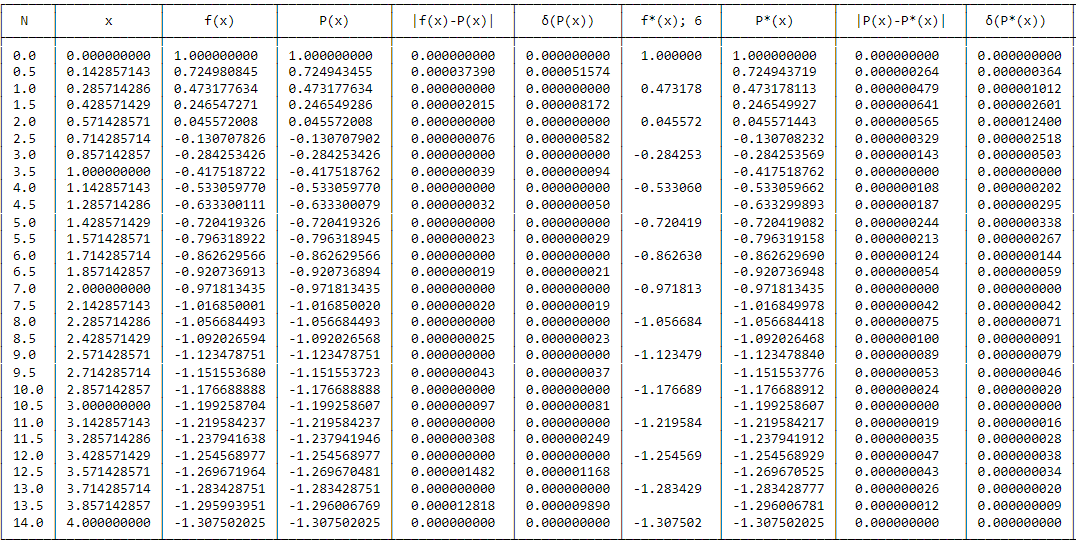
1. Взятие некоторого интервала из области определения функции, на котором будет применена интерполяция.
2. Разбить интервал на некоторое количество значений, взятых с определённым равным шагом. Найти значения функций в этих точках.
3. Построим таблицу разностей, зная значения функции в точках.
4. Используя интерполяционную формулу Ньютона для интерполирования вперёд, найдём значения интерполяционного многочлена в выбранных точках. Сравним значения с изначальной функцией; внесём погрешности в исходные данные и исследуем их влияние на конечные результаты.

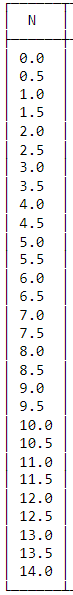
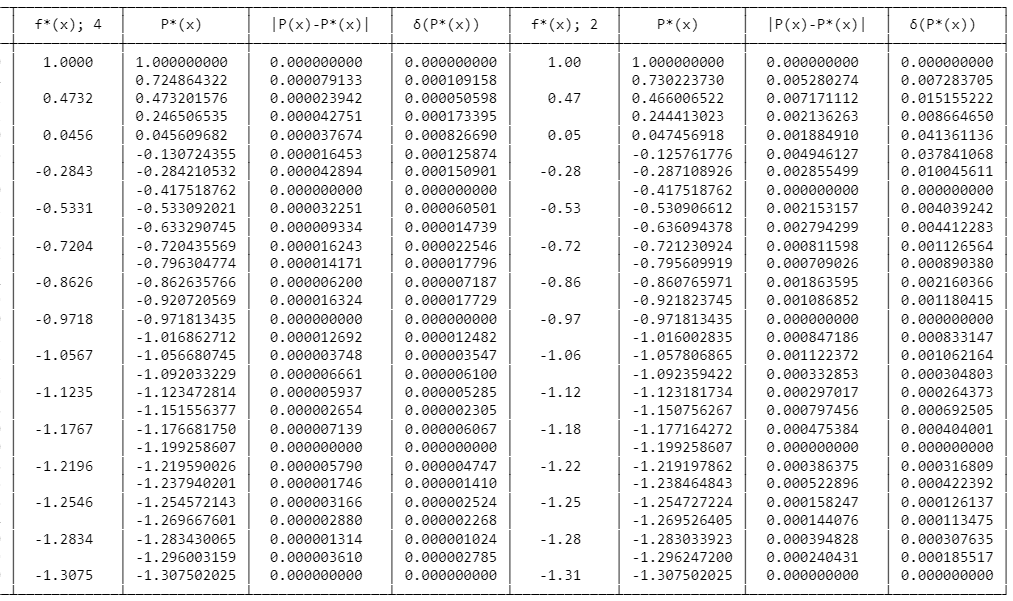
**Выполнение работы:**

1. Дан график функции: . Выберем интервал интерполяции: .
2. Разобьём интервал и найдём значения функции в этих точках.
3. Построим таблицу разностей.
4. Найдём значения интерполяционных многочленов для различных погрешностей.





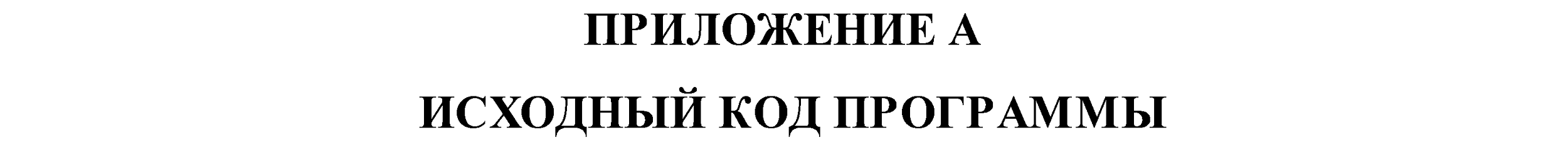




**Вывод:**

Был изучен метод интерполяции многочленами с использованием формулы Ньютона для равноотстоящих узлов. Метод был реализован с использованием языка Python.

Исследованы результаты интерполяции при различных погрешностях: точность метода достаточно высока даже при вполне высоких погрешностях.



from tabulate import tabulate

import numpy as np

import math

node\_arrX = []

n = 15

a = 0

b = 4

filename = "out.txt"

def function(x):

return math.exp(-x) - np.arctan(x)

h = (b - a) / (n - 1)

x\_list: list[float] = [a + i \* h for i in range(n)]

xe\_list: list[float] = []

for i in range(n - 1):

xe\_list.append(x\_list[i])

xe\_list.append((x\_list[i] + x\_list[i + 1]) / 2)

xe\_list.append(x\_list[n - 1])

y\_list: list[float] = [function(x) for x in x\_list]

ye\_list: list[float] = [function(xe) for xe in xe\_list]

# noinspection PyTypeChecker

function\_table = tabulate([["y"] + y\_list, ["x"] + x\_list],

tablefmt="simple\_outline", numalign="center", floatfmt=".9f")

print(function\_table)

with open(filename, "w", encoding="utf-8") as f:

f.write(function\_table)

f.write("\n\n")

class Calculations:

def \_\_init\_\_(self, eps: int = 12):

self.eps = eps

self.x\_list = [round(x, eps) for x in x\_list]

self.xe\_list = [round(xe, eps) for xe in xe\_list]

self.y\_list = [round(y, eps) for y in y\_list]

self.ye\_list = [round(ye, eps) for ye in ye\_list]

self.delta\_matrix = self.get\_delta\_matrix(y\_list)

self.delta\_list = self.delta\_matrix[0]

self.interpolation\_list = self.get\_interpolation\_list(self.xe\_list, self.y\_list, self.x\_list, self.delta\_list)

self.delta\_interpolation = self.get\_delta\_interpolation(self.interpolation\_list, self.ye\_list)

@staticmethod

def get\_delta\_matrix(data\_list: list[float]) -> list[list[float]]:

dim: int = len(data\_list)

delta\_matrix: list[list[float]] = [[0.0] \* dim for \_ in range(dim)]

for i in range(dim):

delta\_matrix[i][0] = data\_list[i]

for i in range(1, dim):

for j in range(dim - i):

delta\_matrix[j][i] = delta\_matrix[j + 1][i - 1] - delta\_matrix[j][i - 1]

return delta\_matrix

@staticmethod

def get\_t\_coefficient(t, num: int):

out = t

for i in range(num, 1, -1):

out \*= t - i + 1

return out

@staticmethod

def get\_interpolation\_list(xe\_data: list[float], y\_data: list[float], x\_data: list[float], delta\_list: list[float]) -> list[float]:

interpolation\_list = []

for x in xe\_data:

result = y\_data[0]

t = (x - x\_data[0]) / h

for i in range(1, len(delta\_list)):

coef = Calculations.get\_t\_coefficient(t, i)

result += coef \* delta\_list[i] / math.factorial(i)

interpolation\_list.append(result)

return interpolation\_list

@staticmethod

def get\_delta\_interpolation(interpolation\_list: list[float], ye\_data: list[float]) -> list[float]:

return [abs(interpolation\_list[i] - ye\_data[i]) for i in range(len(interpolation\_list))]

def print\_delta\_matrix(calc: Calculations) -> None:

table = calc.delta\_matrix

table = np.insert(table, 0, calc.x\_list, axis=1)

headers = ["x", "f"]

headers.extend([f"f{i}" for i in range(len(calc.delta\_matrix) - 1)])

delta\_table = tabulate(table, tablefmt="simple\_outline", headers=headers, numalign="center", floatfmt="0.9f")

print(delta\_table)

with open(filename, "a", encoding="utf-8") as f\_delta:

f\_delta.write(delta\_table)

f\_delta.write("\n\n")

def get\_column(data: list, eps=9) -> list:

column = []

for i in data:

i\_str = format(i, f".{eps}f")

column.extend([i\_str, " "])

return column

def print\_calculations(calc: Calculations, ce\_list:list[Calculations]) -> None:

ie\_list = [i / 2 for i in range(n \* 2)]

del ie\_list[-1]

table = [ie\_list, xe\_list, calc.ye\_list, calc.interpolation\_list,

[abs(calc.interpolation\_list[i] - calc.ye\_list[i]) for i in range(len(ie\_list))]]

for i in range(len(ce\_list)):

table.append(get\_column(ce\_list[i].y\_list, ce\_list[i].eps))

table.append(ce\_list[i].interpolation\_list)

table.append([abs(ce\_list[i].interpolation\_list[j] - calc.interpolation\_list[j]) for j in range(len(ie\_list))])

table = zip(\*table)

headers = ["N", "x", "f(x)", "P(x)", "|f(x)-P(x)|"]

floatfmt = [""]

floatfmt.extend([".9f" for \_ in range(4 + len(ce\_list) \* 3)])

for i in range(len(ce\_list)):

headers.extend([f"f\*(x); {ce\_list[i].eps}", "P\*(x)", "|P(x)-P\*(x)|"])

tab\_table = tabulate(table, tablefmt="simple\_outline", headers=headers, stralign="center",

numalign="center", floatfmt=floatfmt)

with open(filename, "a", encoding="utf-8") as f\_calc:

f\_calc.write(tab\_table)

f\_calc.write("\n\n")

print(tab\_table)

def main() -> None:

calc: Calculations = Calculations()

print\_delta\_matrix(calc)

ce\_list: list[Calculations] = []

for i in [6, 4, 2]:

ce\_list.append(Calculations(i))

print\_calculations(calc, ce\_list)

if \_\_name\_\_ == "\_\_main\_\_":

main()