

# Отчёт по лабораторной работе №1. Поповкин Артемий. Б9122-02.03.01 СЦТ.

## Вариант-15

### Цель

1. Построить интерполяционный многочлен Лагранжа для заданной функции
2. Построить таблицу абсолютной, относительной и теоретической погрешностей для каждого  $n$
3. Построить график зависимостей  $\Delta f_n(n)$ ,  $r_n(n)$
4. Сделать вывод.

### Условия

Промежуток:  $[0.1, 0.6]$

$$f(x) = 2x - \cos(x)$$

$n \in [3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]$

### Ход работы

#### Подготовительный этап

#### Используемые библиотеки

В ходе работы мне потребовалось использовать следующие библиотеки:

```
from numpy import cos, linspace, pi
from math import factorial
import pandas as pd
"""Для хранения данных"""
import matplotlib.pyplot as plt
"""
Для вырисовывания графика функции.
```

Все остальные графики нарисованы с помощью pandas  
"""

Все они, за исключением **matplotlib**, активно использовались для решения ряда задач.

**numpy** и **math** позволили вычислять теоретическую ошибку и саму функцию, а **linspace** позволил генерировать равноудалённые точки на промежутке.

**pandas** был использован для структуризации данных в конце. Также с его помощью были произведены графики ошибок.

**matplotlib** потребовался для отрисовывания графика функции на промежутке.

## Определение функции и её производных

```
def func(x: float):  
    """  
    Функция  $y = 2x - \cos(x)$   
  
    :param x: Точка x  
    :return: Значение функции в точке x  
  
    :rtype: float  
  
    """  
    return 2 * x - cos(x)
```

```
def derv_func(x: float, n: int = 2) -> float:  
    """  
    Вторая-n производная функции  $y = 2x - \cos(x)$   
  
    От второй производной далее, по тригонометрическим свойствам, можно считать n-ю производную  
  
    :param x: Точка x  
    :param n: Степень производной. Считается от 2 для удобства.  
        + Потому что именно на 2-й производной убирается константа  
    :return: Значение функции в точке x  
  
    :rtype: float  
  
    """  
    return cos(x + ((n - 2) * pi) / 2)
```

```
rng_pos = [3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
"""Количество точек"""
range_graph = (0.1, 0.6)
"""Промежуток"""
```

Все функции были закодированы с использованием *numpy*.

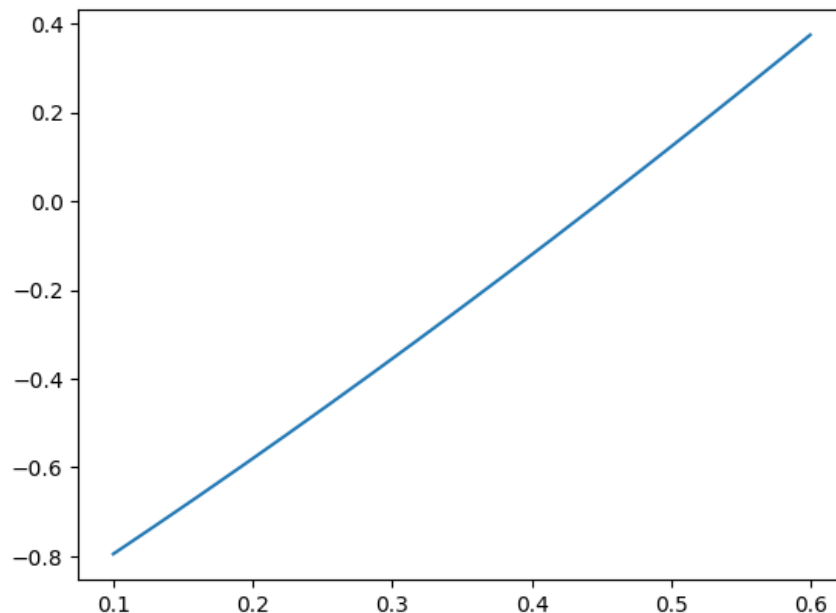
В вычислении производной берётся вторая производная, так как начиная с неё остаётся лишь  $\cos(x + \frac{(n-2) \cdot \pi}{2})$ , что при взятии производной переходит на  $\frac{\pi}{2}$  по единичной окружности.

Графическое доказательство.

## Отображение функции

```
import matplotlib.pyplot as plt

plt.plot(func(linspace(*range_graph, num=10 ** 3)))
```



## Вычисление полинома Лагранжа

### Генерация точек

```
def generate_points(rng: tuple[float, float], count_points: int, function) -
> list[tuple[float, float]]:
    """
```

Генерирование точек с некоторым постоянным шагом

```
:param rng: Кортеж границ
:type rng: tuple[float, float]
:param count_points: Количество точек, которое нужно сгенерировать
:type count_points: int
:param function: Функция
:type function: function

:return: Список точек
:rtype: list[tuple[float, float]]

"""
res = []
for pos in linspace(*rng, count_points):
    res.append((pos, function(pos)))
return res
```

Как было сказано выше, помощью **linspace** можно “раздробить” пространство в заданном диапазоне на нужное количество точек с равным шагом.

Тогда, по необходимости, можно дробить промежуток на нужное количество точек..

## Определение функции полинома Лагранжа

По формуле, через цикл, умножаем функцию в точке  $k$  на произведение дробей, исключая при этом саму точку  $k$ .

```
def lagrange(bp: float, points: list[tuple[float, float]]) -> float:
    """
    Неньютоновская реализация полинома Лагранжа

    :param points: Список точек
    :type points: list[tuple[float, float]]
    :param bp: Точка, значение функции в которой нужно получить
    :type bp: float

    :return: Значение полинома Лагранжа в точке bp
    :rtype: float

    """
    count_points = len(points)
    result = 0
    for k, point in enumerate(points):
        multiply = point[1]
```

```

    for j in range(0, k - 1 + 1):
        x = points[j][0]
        multiply *= ((bp - x) / (point[0] - x))
    for i in range(k + 1, count_points):
        x = points[i][0]
        multiply *= ((bp - x) / (point[0] - x))
    result += multiply
return result

```

Графическое доказательство.

## Вычисление ошибок

### Норма

Берётся по-формуле.

Сейчас и далее будет браться 1000 точек на промежутке.

Это число выбрано как баланс между точностью на сравнительно малом отрезке и временем компиляции

```

def get_norm(function, rng: tuple[float, float], *args) -> float:
    """
    Получение нормы функции

    :param function: Некоторая функция, норму которой мы хотим получить
    :param rng: Кортеж границ
    :param args: Дополнительные аргументы функции

    :return: Норма функции
    :rtype: float

    """
    return max(abs(function(linspace(*rng, num=10 ** 3), *args)))

```

### Ошибки

В силу особенностей реализации функции взятия нормы, для получения абсолютной ошибки необходимо проводить операцию взятия нормы вне функции.

```

def rel_error(abs_er: float, norm_f: float) -> float:
    """
    Получение относительной ошибки

    :param abs_er: Абсолютная ошибка

```

```

:param norm_f: Норма функции

:return: Относительная ошибка
:rtype: float

"""
return (abs_er / norm_f) * 100

def teor_error(count_points: int, rng: tuple[float, float]) -> float:
    """
    Получение теоретической ошибки

    :param count_points: Количество точек
    :param rng: Кортеж границ

    :return: Теоретическая ошибка
    :rtype: float

    """
    return (get_norm(derv_func, rng, count_points + 1) / factorial(count_points + 1)) * (
        (rng[1] - rng[0]) ** (count_points + 1))

abs_e = max(abs(lagrange(linspace(*range_graph, num=10 ** 3), full_points) -
func(linspace(*range_graph, num=10 ** 3))))
"""Абсолютная ошибка"""

```

## Основной цикл

Цикл перебирает все количества точек из `rng_pos`.

```

abs_e_mass = []
rel_e_mass = []
ter_e_mass = []

for count_pts in rng_pos:
    full_points = generate_points(range_graph, count_pts, func)

    norm = get_norm(func, range_graph)
    lag_norm = get_norm(lagrange, range_graph, full_points)
    abs_e = max(abs(lagrange(linspace(*range_graph, num=10 ** 3), full_points) -
func(linspace(*range_graph, num=10 ** 3))))

```

```

der_e = get_norm(derv_func, range_graph)
rel_e = rel_error(abs_e, lag_norm)
ter_e = teor_error(count_pts, range_graph)

abs_e_mass.append(abs_e)
rel_e_mass.append(rel_e)
ter_e_mass.append(ter_e)

abs_df = pd.DataFrame(abs_e_mass, rng_pos, columns=["Value"])
rel_df = pd.DataFrame(rel_e_mass, rng_pos, columns=["Value"])
ter_df = pd.DataFrame(ter_e_mass, rng_pos, columns=["Value"])

total_df = pd.DataFrame({"Абсолютная ошибка": abs_e_mass,
                        "Относительная ошибка": rel_e_mass,
                        "Теоретическая ошибка": ter_e_mass},
                        index=rng_pos)
total_df.to_csv("Все значения.csv")

```

`*_df` — сохранение данных для более удобного перевода в графики и ручной оценки ошибок.

## Таблица с данными

В результате проведения вычислений, был получен `DataFrame` `total_df`, содержащий в каждой строке количество точек, а также значения ошибок при этом количестве точек.

В силу технических обстоятельств, программа, в которой я пишу отчёт, Notion, позволяет корректно отображать `.csv` только при наличии заголовка строки, коим в данном случае является первый столбец.

### Значения

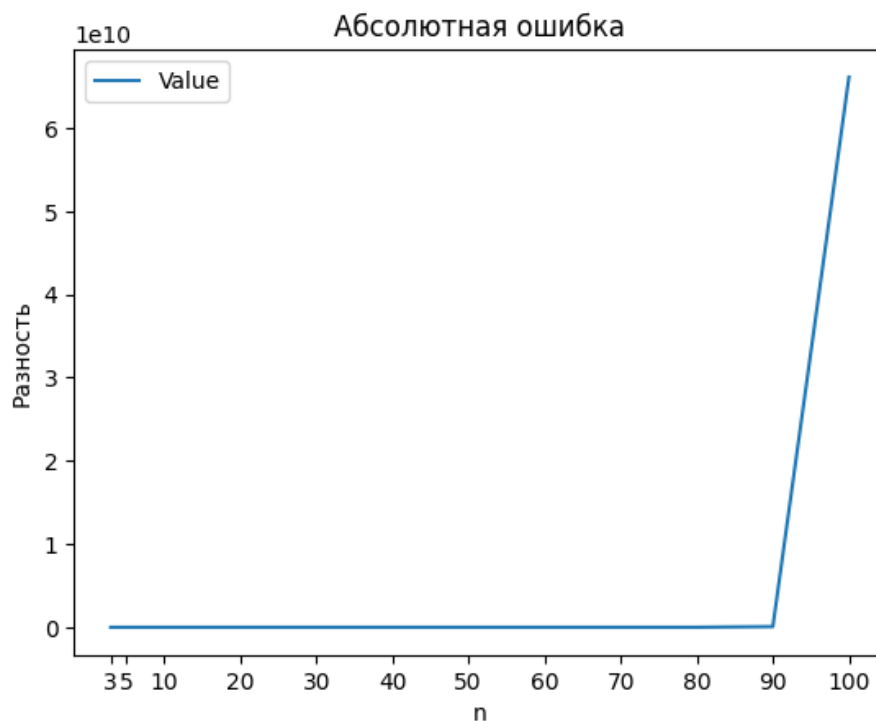
Aa Property	# Количество точек	# Абсолютная ошибка	# Относительная ошибка	# Теоретическая ошибка
<u>A</u>	3	0.0003766333098933622	0.047375010892131295	0.0025911566804115256
<u>B</u>	5	0.0000003	0.00004345395794646543	0.000021592972336762714
<u>C</u>	10	0.0000000000000005	0.000000000000006	0.00000000000007
<u>D</u>	20	0.00000000000004	0.000000000005	5.269867827247845e-27
<u>E</u>	30	0.0000000002	0.00000002	3.197583609355706e-44
<u>F</u>	40	0.0000002	0.000019854888466317122	7.675643515468616e-63
<u>G</u>	50	0.00013561493847147332	0.017058393451818782	1.616585637471971e-82
<u>H</u>	60	0.0949788344165754	11.290856338358376	4.82435350784224e-103
<u>I</u>	70	47.48382435099228	101.69131984828093	2.8117743633623897e-124
<u>J</u>	80	57361.4663247959	100.00138047564458	4.028386698381146e-146

Aa Property	# Количество точек	# Абсолютная ошибка	# Относительная ошибка	# Теоретическая ошибка
<u>K</u>	90	77955310.00554986	99.99999952267834	1.6868122864342624e-168
<u>L</u>	100	66128295809.75994	100.00000000119744	2.3627564253351933e-191

## Построение графиков зависимостей

### Абсолютная ошибка

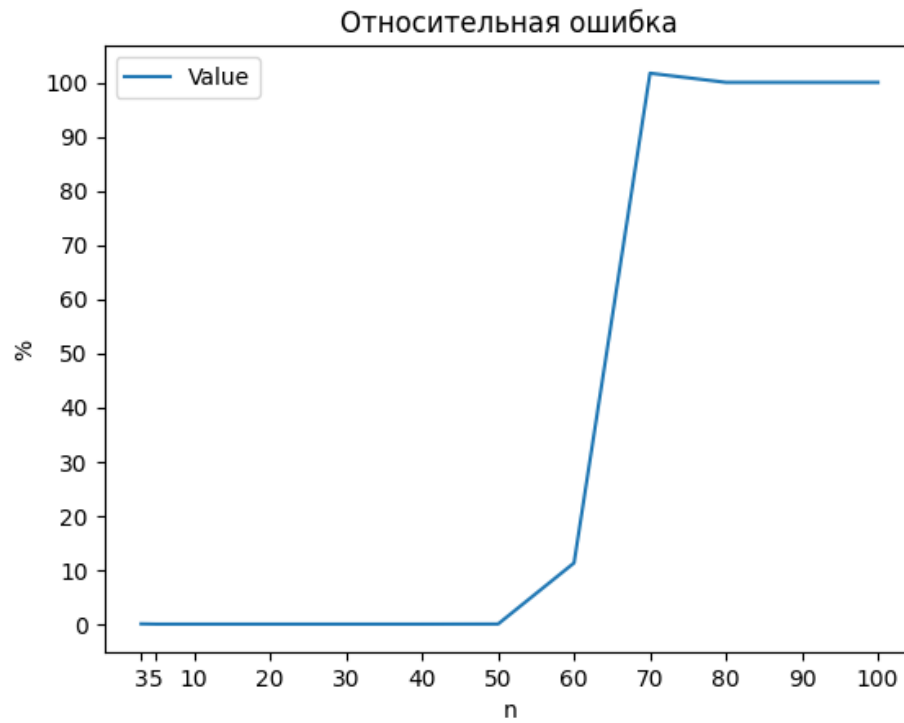
```
abs_df.plot(title="Абсолютная ошибка", xticks=rng_pos, xlabel="n", ylabel="Разность")
```



### Относительная ошибка

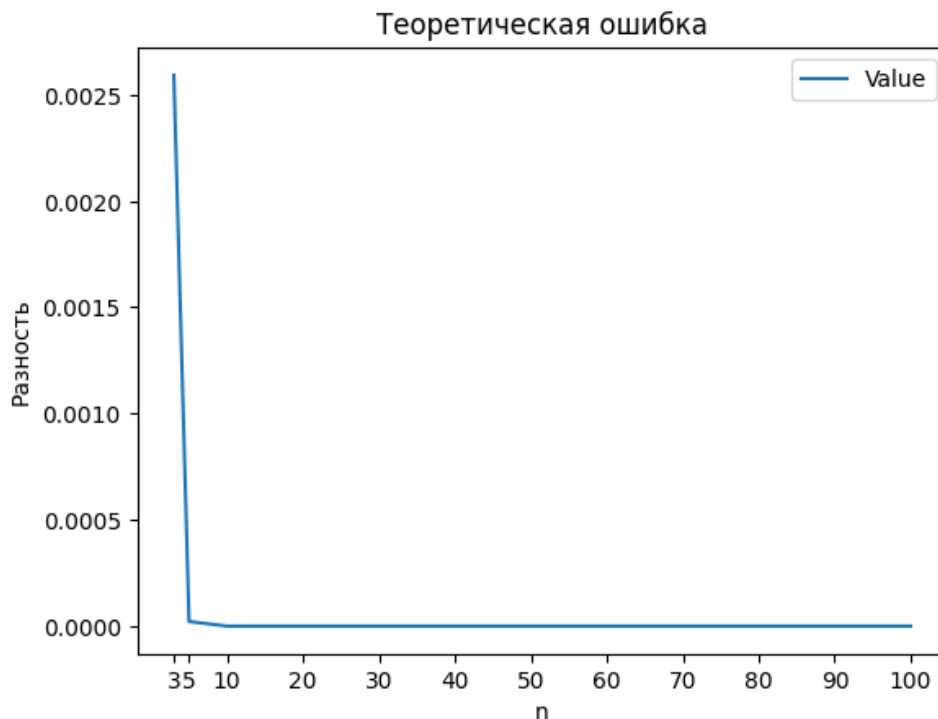
```
rel_df.plot(title="Относительная ошибка", xticks=rng_pos, yticks=range(0, 100 + 1, 10), xlabel="n", ylabel="%")
```





### Теоретическая ошибка

```
ter_df.plot(title="Теоретическая ошибка", xticks=rng_pos, xlabel="n", ylabel="Разность")
```



## Вывод

В результате исследования можно заметить, что с увеличением количества точек абсолютная ошибка стремится к бесконечности, относительная стремится к 100%, а теоретическая, наоборот, стремится к 0.

Постараемся проанализировать и понять, почему получаются именно такие результаты.

Можно заметить, что до 10-ти точек абсолютная ошибка стремилась к 0, но после 10 она начала стремиться к  $\infty$ .

Дело заключается в множителе Лагранжа:

$$\frac{bp - \text{points}[j][0]}{\text{point}[0] - \text{points}[j][0]}$$

С увеличением числа точек, расстояние между координатами  $\text{point}[0]$  и  $\text{points}[j][0]$  стремится к 0, что и порождает данный эффект.

По этой же причине относительная ошибка стремится к 100%.

Перейдём к тому, почему теоретическая ошибка стремится к нулю.

Поскольку функция тригонометрическая (Начиная со 2-й производной, она отбрасывает все лишние члены, становясь полностью тригонометрической), то она принимает в точке  $x = n + 1$  производную, что говорит нам о том, что функция раскладывается в ряд Тейлора, а значит остаточный член, отвечающий за теоретическую погрешность, стремится к 0.


Ссылка на полную версию кода ( [Jupyter Notebook](#) ):

CalcMath/Lagrange/lagrange.ipynb at main · Jrol123/CalcMath

Contribute to Jrol123/CalcMath development by creating an account on GitHub.

<https://github.com/Jrol123/CalcMath/blob/main/Lagrange/lagrange.ipynb>

Jrol123/CalcMath



AK 1

Contributor

0

Issues

0

Stars

0

Forks

