

Санкт-Петербургский политехнический университет Петра Великого  
Институт компьютерных наук и технологий  
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

## **Отчёт по лабораторной работе № 1**

Дисциплина: Вычислительная математика

Выполнил студент гр. 3530901/10001 \_\_\_\_\_ Д.Л. Симоновский  
(подпись)

Руководитель \_\_\_\_\_ В.Н. Цыган  
(подпись)

“09” февраль 2023 г.

Санкт-Петербург

2023

## Оглавление

<b>Задание:</b> .....	<b>2</b>
<b>Инструменты:</b> .....	<b>2</b>
<b>Ход выполнения работы:</b> .....	<b>2</b>
<i>Порядок действий:</i> .....	2
<i>Первая задача:</i> .....	2
Полная формулировка: .....	2
Разбиваем на задачи: .....	2
Пункт 1: .....	2
Пункт 2: .....	3
Пункт 3: .....	3
<i>Вторая задача:</i> .....	5
Полная формулировка: .....	5
Проблемы: .....	5
Решение: .....	5
<b>Вывод:</b> .....	<b>7</b>
<b>Ссылки:</b> .....	<b>8</b>

## Задание:

Вариант 11:

Для функции  $f(x) = \frac{1}{1+x}$  по узлам  $x_k = 0.1k$  ( $k = 0, 1, \dots, 10$ ) построить полином Лагранжа  $L(x)$  10-й степени и сплайн-функцию  $S(x)$ . Вычислить значения всех трех функций в точках  $y_k = 0.05 + 0.1k$  ( $k = 0, 1, \dots, 9$ ). Результаты отобразить графически.

Используя программу QUANC8, вычислить два интеграла:  $\int_2^5 (abs(x - tg(x)))^m dx$ , для  $m = -1$  и для  $m = -0.5$ .

## Инструменты:

Для работы был выбран язык программирования Python версии 3.11 по причине удобства его использования для поставленной задачи. Были выбраны следующие библиотеки:

- NumPy – для большей скорости расчетов
- SciPy – для функций расчета интерполяции и интеграла
- PrettyTable – для красивого вывода в консоль таблицы
- Matplotlib – для вывода графиков

## Ход выполнения работы:

### Порядок действий:

Поставленное задание легко можно разбить на две глобальные задачи:

1. Выполнить интерполяцию  $f(x) = \frac{1}{1+x}$
2. Выполнить расчет интеграла  $\int_2^5 (abs(x - tg(x)))^m dx$  при  $m = -1, -0.5$

### Первая задача:

Полная формулировка:

Для функции  $f(x) = \frac{1}{1+x}$  по узлам  $x_k = 0.1k$  ( $k = 0, 1, \dots, 10$ ) построить полином Лагранжа  $L(x)$  10-й степени и сплайн-функцию  $S(x)$ . Вычислить значения всех трех функций в точках  $y_k = 0.05 + 0.1k$  ( $k = 0, 1, \dots, 9$ ). Результаты отобразить графически.

Разбиваем на задачи:

В поставленной задаче явно есть несколько этапов:

1. Построить полином Лагранжа и сплайн-функцию
2. Вычислить значения в точках  $y_k = 0.05 + 0.1k$  ( $k = 0, 1, \dots, 9$ )
3. Вывести результаты графически

Пункт 1:

Создадим узлы интерполяции. Для этого воспользуемся функцией NumPy, который при добавлении мы переименовали в np для удобства:

```
x_interpolation = np.arange(0, 1 + 0.1, 0.1)
y_interpolation = 1 / (1 + x_interpolation)
```

Далее выполним интерполяцию, используя функции библиотеки SciPy:

```
lagrange_interpolation = lagrange(x_interpolation, y_interpolation)
spline_interpolation = CubicSpline(x_interpolation, y_interpolation)
```

## Пункт 2:

Аналогично узлам интерполяции создадим массив узлов, в которых будем считать погрешность интерполирования и выводить графики:

```
x_find = np.arange(0.05, 1, 0.1)
```

Далее необходимо найти значения полиномов в точках `x_find`, для этого выполним:

```
y_lagrange = lagrange_interpolation(x_find)
y_spline = spline_interpolation(x_find)
```

## Пункт 3:

Для отрисовки рассчитаем реальное значение функции в этой точке:

```
y_real = 1 / (1 + x_find)
```

Далее, в традициях функционального программирования, напомним функцию для отрисовки значений полиномов.

Напишем вывод полученных значений в консоль, используя библиотеку PrettyTable:

```
def print_interpolation_table(x_find, y_real, y_lagrange, y_spline):
    pt = PrettyTable()
    pt.add_column('x', np.round(x_find, 4))
    pt.add_column('real y', y_real)
    pt.add_column('lagrange y', y_lagrange)
    pt.add_column('spline y', y_spline)
    print(pt)
```

Полученный вывод:

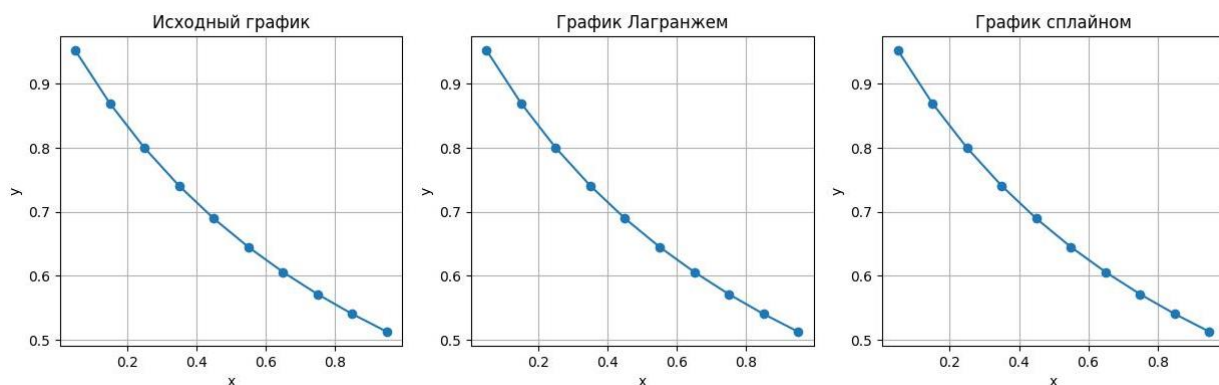
x	real y	lagrange y	spline y
0.05	0.9523809523809523	0.952380997794005	0.9524187787963349
0.15	0.8695652173913044	0.8695652108444114	0.869550918173362
0.25	0.8	0.8000000017723101	0.8000010916337602
0.35	0.7407407407407407	0.7407407399777892	0.7407385880854702
0.45	0.689655172413793	0.6896551729111877	0.6896544461342486
0.55	0.6451612903225805	0.6451612898606283	0.6451605687694763
0.65	0.606060606060606	0.6060606066632109	0.606060071504933
0.75	0.5714285714285714	0.5714285700738855	0.5714283935768044
0.85	0.5405405405405405	0.5405405443786875	0.5405395206825169
0.95	0.5128205128205128	0.5128204878755788	0.5128230524168983

Далее по заданию необходимо вывести графики. Сделаем это используя библиотеку Matplotlib, а конкретно модуль pyplot, переименуем его в plt.

```
def print_one_graph(x, y, title, id, count_graphs):
    plt.subplot(1, count_graphs, id)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.title(title)
    plt.plot(x, y, '-o')

def print_interpolation_graph(x_find, y_real, y_lagrange, y_spline):
    plt.figure(figsize=(15, 4))
    print_one_graph(x_find, y_real, 'Исходный график', 1, 3)
    print_one_graph(x_find, y_lagrange, 'График Лагранжем', 2, 3)
    print_one_graph(x_find, y_spline, 'График сплайном', 3, 3)
    plt.savefig("Graphs.jpg")
    plt.show()
```

Полученные графики:

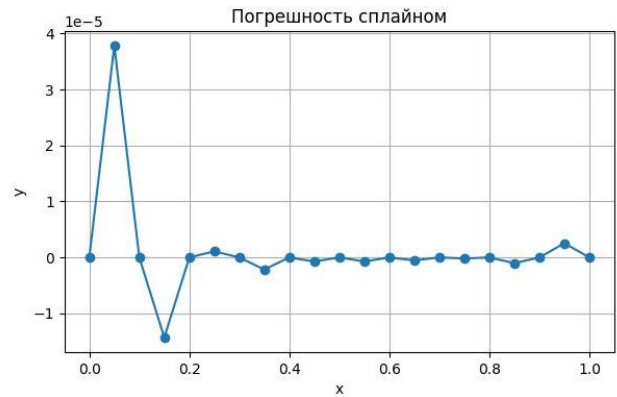
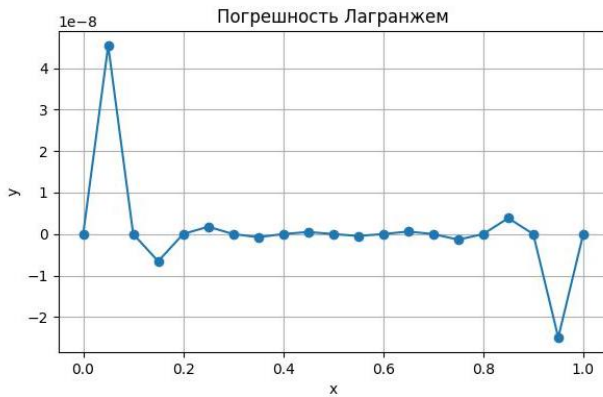


Графики. Graphs.jpg

Выведем график погрешности т.к. на самих графиках не видно разницы. Будем использовать те же модули:

```
def print_interpolation_error(x_interpolation, x_find, y_lagrange_error, y_spline_error):
    plt.figure(figsize=(15, 4))
    # Добавляем узлы т.к. при интерполяции в них погрешность нулевая
    x = np.arange(0, 1 + 0.05, 0.05)
    # Добавляем нулевые узлы на график
    y_lagrange = [0 if i % 2 == 0 else y_lagrange_error[i // 2] for i in range(0,
len(x_interpolation) + len(x_find))]
    y_spline = [0 if i % 2 == 0 else y_spline_error[i // 2] for i in range(0,
len(x_interpolation) + len(x_find))]
    # Собственно сам график
    print_one_graph(x, y_lagrange, 'Погрешность Лагранжем', 1, 2)
    print_one_graph(x, y_spline, 'Погрешность сплайном', 2, 2)
    plt.savefig("Error.jpg")
    plt.show()
```

Полученные графики:



Графики погрешности. Error.jpg

## Вторая задача:

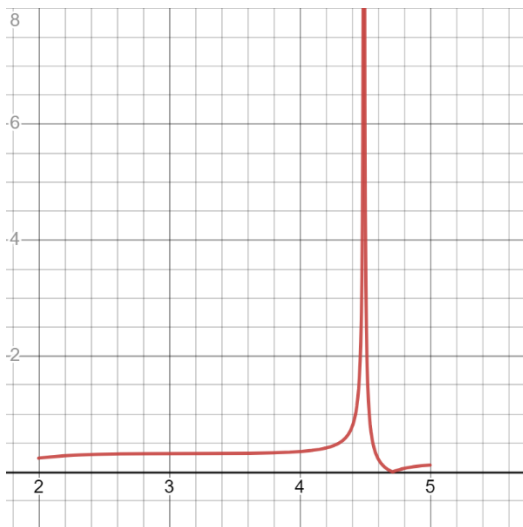
Полная формулировка:

Используя программу QUANC8, вычислить два интеграла:  $\int_2^5 (abs(x - tg(x)))^m dx$ , для  $m = -1$  и для  $m = -0.5$ .

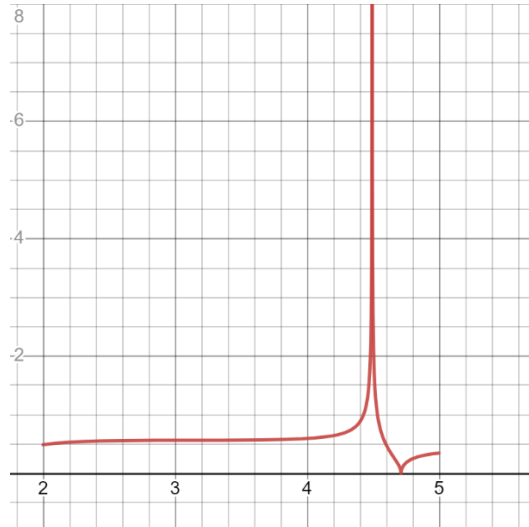
Проблемы:

Построим график подынтегральной функции в сторонних приложениях.

При  $m = -1$ :



При  $m = -0.5$



Как видно из графиков у нас имеется точка, в которой функция уходит в бесконечность, её значение найдем путем решения уравнения  $tg(x) = x$ , эта точка  $x = \sim 4.49340945790906$ .

При решении придется её отдельно обработать.

Решение:

Создаем функцию, которая будет на основании  $m$  возвращать нам другую функцию для интегрирования:

```
def get_integrate_function(m):  
    def integrate_function(x):  
        return (np.abs(x - np.tan(x))) ** m  
  
    return integrate_function
```

Так же т.к. в Python нет реализации QUANC8, возьмем аналог, основанный на адаптивных квадратурных формах, опирающийся на методы из библиотеки QUADPACK, языка Fortran:

```
def pseudo_quanc8(func, start, end):  
    return quad(func, start, end, limit=30)
```

Используя эти функции напомним функцию для интегрирования нашего выражения:

```
def do_integrate(approximation, integrate_function):  
    inconvenient_point = 4.49340945790906  
    integral = pseudo_quanc8(integrate_function, 2, inconvenient_point - approximation) + \  
        pseudo_quanc8(integrate_function, inconvenient_point + approximation, 5)  
    return integral[0]
```

Для получения итогового ответа необходимо найти константу, на которую мы хотим приблизиться к месту, где подынтегральная функция уходит в бесконечность (в коде – approximation). Для этого будем вызывать функцию do\_integrate несколько раз с разной константой:

```
def do_integration_with_various_approximations(approximation_array, m):  
    """Эта функция считает значение интеграла с различным приближением к точке разрыва"""  
    integrate_function = get_integrate_function(m)  
    integration_return = []  
    for i in approximation_array:  
        integration_return.append(do_integrate(i, integrate_function))  
    return integration_return
```

Вызовем эту функцию для разных значений m и выведем полученное значение на экран. Для вывода используем уже знакомую библиотеку PrettyTable:

```
def print_integration_table(approximation, array_with_m_05, array_with_m_1):  
    pt = PrettyTable()  
    approximation_decimal = [f'{i:.0e}' for i in approximation]  
    pt.add_column('Approximation', approximation_decimal)  
    pt.add_column('Integral, m=-0.5', array_with_m_05)  
    pt.add_column('Integral, m=-1.0', array_with_m_1)  
    print(pt)  
  
def integration():  
    approximation_array = [10 ** (-i) for i in range(20)]  
    print_integration_table(  
        approximation_array,  
        do_integration_with_various_approximations(approximation_array, -0.5),  
        do_integration_with_various_approximations(approximation_array, -1.0)  
    )
```

Полученный вывод:

Approximation	Integral, m=-0.5	Integral, m=-1.0
1e+00	0.8246885157562769	0.4559263679289679
1e-01	1.3768735556532714	0.7982425929285323
1e-02	1.4832422904082658	0.9334784353723861
1e-03	1.5140133996482101	1.049623990263057
1e-04	1.5236483242172452	1.1638759249932669
1e-05	1.5266921080044722	1.2779386506567976
1e-06	1.5276545405394897	1.391982456775419
1e-07	1.5279588853791268	1.5060243691891981
1e-08	1.5280996361255148	1.6200660732222396
1e-09	1.5280996377834404	1.7341075800570158
1e-10	1.5280996371891238	1.8481472984740965
1e-11	1.5280996308901285	1.96216940852405
1e-12	1.5280996368966167	2.076011739320603
1e-13	1.5280996371756375	2.1880509246849824
1e-14	1.528099637198792	2.2882942941577844
1e-15	1.528099637200216	2.339310710756356
1e-16	1.5280996372072886	2.3451523327137584
1e-17	1.5280996372072886	2.3451523327137584
1e-18	1.5280996372072886	2.3451523327137584
1e-19	1.5280996372072886	2.3451523327137584

Однако помимо самой таблицы также было получено 2 предупреждения:

```
...\main.py:83: IntegrationWarning: The maximum number of subdivisions (50) has been achieved.
```

```
If increasing the limit yields no improvement it is advised to analyze the integrand in order to determine the difficulties. If the position of a local difficulty can be determined (singularity, discontinuity) one will probably gain from splitting up the interval and calling the integrator on the subranges. Perhaps a special-purpose integrator should be used.
```

```
...\main.py:83: IntegrationWarning: Extremely bad integrand behavior occurs at some points of the integration interval.
```

Первое сообщение появляется, когда функция для вычисления интеграла первый раз обрубает количество делений промежутка на 30 (в сообщении об ошибке указано число 50, однако при вызове мы устанавливаем лимит 30, вероятно это какой-то баг в используемой библиотеке).

Второе сообщение просит нас проверить, а все ли хорошо с функцией, интеграл которой мы вычисляем. Оно появляется т.к. мы достигли значения максимального числа вычислений.

Этот момент можно заметить в таблице с результатами интегрирования, а именно по одинаковым результатам вычислений при приближении 1e-16, 1e-17, 1e-18, 1e-19. Это связано с тем, что в отличие от первого предупреждения, второе останавливает вычисления и выводит тот ответ, который у неё получился на текущем этапе.

## Вывод:

В ходе работы я ознакомился с аналогами QUANC8, SPLINE, SEVAL на языке Python и получил опыт работы с ними, так же научился обрабатывать исключительные ситуации, как во втором задании. По результатам работы видно, что сплайн-функция приближает много лучше, чем полином Лагранжа, а также видно, что интерполяция тем лучше, чем дальше значение от краёв промежутка.



## **Ссылки:**

С листингом кода можно ознакомиться на [github.com](https://github.com)