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

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, напишем 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:

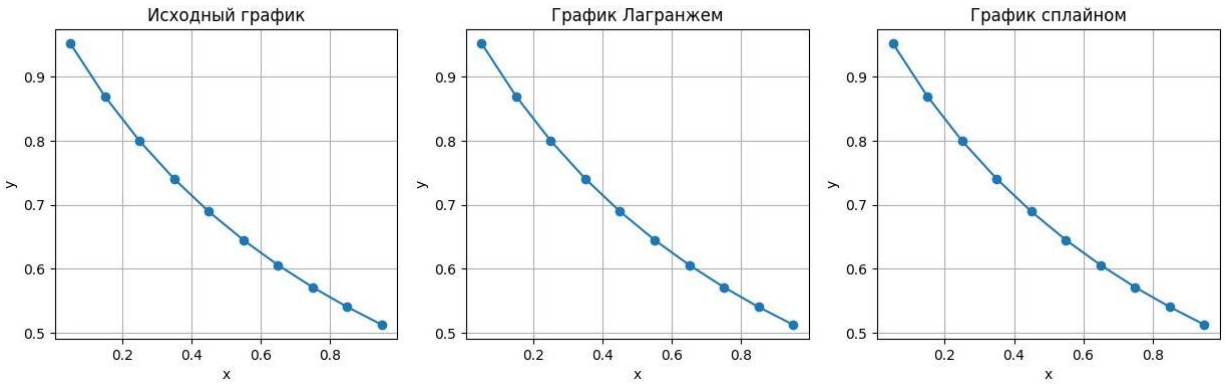
```
+------+--------------------+--------------------+--------------------+
|  x   |       real y       |     lagrange y     |      spline y      |
+------+--------------------+--------------------+--------------------+
| 0.05 | 0.9523809523809523 | 0.952380997794005  | 0.9524187787963349 |
| 0.15 | 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:

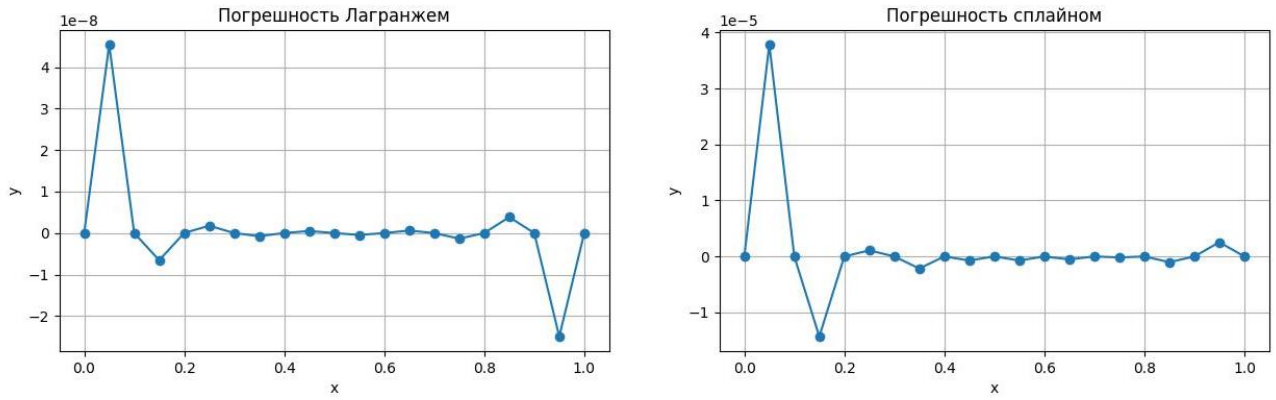


Графики. Graphs.jpg

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:

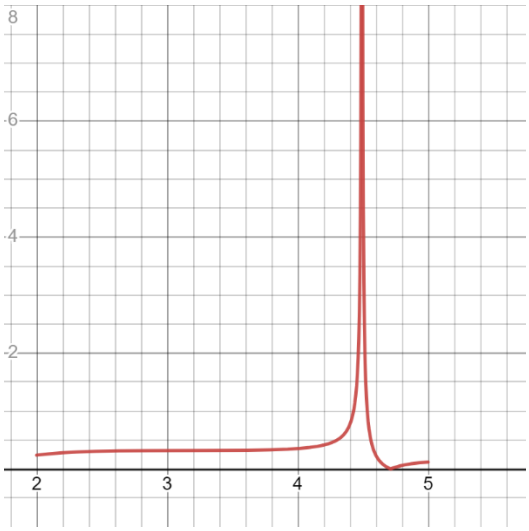
Полная формулировка:

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

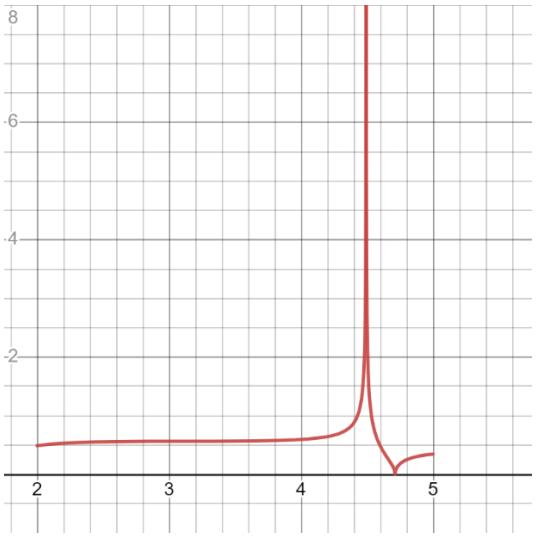
Проблемы:

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 = ~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 напишем функцию для интегрирования нашего выражения:

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

Для получения значения интегралов вызовем этот метод 2 раза и выведем значения на экран:

```
def integration():
    print(f'Integral with m=-1: {do_integrate(-1.0, 2e-9)}')
    print(f'Integral with m=-0.5: {do_integrate(-0.5, 1e-6)}')
```

Получим
Integral with m=-1: 1.6997777239732015
Integral with m=-0.5: 1.5276545405394897
Однако эти значения являются примерными т.к. мы не имеем возможности посчитать бесконечное значение у графика, точность, насколько мы приближаемся к пику мы высчитываем путем перебора значений приближения.

Вывод:

В ходе работы я ознакомился с аналогами QUANC8, SPLINE, SEVAL на языке Python и получил опыт работы с ними, так же научились 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