Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и супуркомпьютерных технологий

Отчет по лабораторной работе № 1

Дисциплина: Вычислительная математика

Выполнил студент гр. 3530901/10003 ____________ Рубцов Е.А.
(подпись)

Руководитель _______________ Цыган В.Н.
(подпись)

“___”_________2023 г.

Санкт-Петербург

2023

Contents

Задание:

Для функции f(x) = $\frac{1}{1+25x^2}$ по узлам x_k = -1 + 0.1k (k=0,1,…20) построить полином Лагранджа L(x) 20-й степени и сплайн функцию S(x). Вычислить значения всех трех функций в точках y_k = -0.95 + 0.1k (k=0,1,…19). Результаты отобразить графически.

Используя программу **QUANC8** вычислить два интеграла:
$\int_0^{2.14} |1 - x^2|\,, dx$, для m = -1 и для m = -0.5

Инструменты:

Для выполнения поставленного задания был выбран язык python по причине удобства его использования и быстроты выполнения поставленной задачи. Были использованны следующие библиотеки:

1. NumPy – эта библиотека предостовляет функции для более быстрого выполнения расчетов
2. SciPy – содержит неообходимые функции для расчета интегралов и построения интерполяционных полиномов
3. MatplotLib – позволяет удобно строить графики и сохранять их в формате изображения
4. Tabulate – для вывода точных значений точек в консоль

Ход выполнения работы:

Задача интерполяции:

Построение полиномов:

Создадим узлы интерполяции(func_x), а так же массив значений функции в данных точках(func_y). Для создания узлов интерполяции используется функция numpy.arange, которая позволяет построить массив равнораспределенных точек в некотором интервале.

```
FUNC = lambda x: 1/(1 + 25 * np.power(x, 2))

func_x = np.arange(X_START, X_START + X_STEP * (X_STEP_COUNT + 1),
X_STEP)
func_y = np.array(list(map(FUNC, func_x)
```

Далее выполним интерполяцию по данным узлам. Для этого используются функции из библиотеки SciPy,

```
lagrange_interpolation = lagrange(func_x, func_y)
spline_interpoation = CubicSpline(func_x, func_y)
```

Получение значений в точках и построение графиков:

Аналогично предидущему пункту создадим массив узлов, в которых будет считаться значения полиномов и их погрешность:

```
x_shifted = n.arange(X_START + 0.05, X_START + 0.05 + X_STEP *
STEP_COUNT, X_STEP)
```

Посчитаем значения полиномов и изначальной функции в этих точках и выведем их в таблицу:

```
def print_table(columns, rows):
    table = []
    for i in range(0, len(rows[0])):
        table.append([j[i] for j in rows])
    print(tabulate(table, columns, tablefmt="pretty"))


print_table(
 ["x", "original", "lagrange", "spline"],
 [
 x_shifted,
 np.array(list(map(FUNC, x_shifted))),
 lagrange_interpolation(x_shifted),
 spline_interpolation(x_shifted)
 ]
)
```

Полученная таблица:

x	original	lagrange	spline
-0.95	0.042440318302387266	-39.95244810844703	0.04245771691214385
-0.85	0.05245901639344263	3.4549580463575036	0.05245178535029959
-0.75	0.06639004149377593	-0.4470519042363712	0.06638856096438338
-0.65	0.08648648648648648	0.20242262664180832	0.08647510286763853
-0.55	0.11678832116788321	0.08065999516671285	0.11678648012734957
-0.45	0.16494845360824736	0.17976263012073435	0.16486466302504504
-0.35	0.24615384615384606	0.23844593375846213	0.24626813037193943
-0.25	0.3902439024390242	0.3950930536889238	0.38941957941292604
-0.15	0.6399999999999997	0.6367553359164506	0.6431689365917397
-0.05	0.941176470588235	0.9424903797439836	0.9388662126816517
0.05	0.9411764705882357	0.9424903797439848	0.9388662126816524
0.15	0.6400000000000009	0.6367553359164329	0.6431689365917411
0.25	0.3902439024390248	0.3950930536877747	0.38941957941292665
0.35	0.2461538461538463	0.23844593373804668	0.24626813037193968
0.45	0.16494845360824759	0.17976262989953207	0.16486466302504527
0.55	0.11678832116788336	0.08065999341256558	0.11678648012734973
0.65	0.08648648648648656	0.20242261564235464	0.08647510286763861
0.75	0.06639004149377598	-0.4470519610778825	0.06638856096438343
0.85	0.052459016393442665	3.454957797987107	0.05245178535029962
0.95	0.04244031830238731	-39.95244904166532	0.0424577169121439

Похожим образом получаем графики полиномов:

```
def draw_plots(plot_values, plot_titles, file_name):
    fig, *axes = plt.subplots(1, len(plot_values))

    for i in range(0, len(plot_values)):
        axes[0][i].title.set_text(plot_titles[i])
        axes[0][i].set(
            xlabel = "x",
            ylabel = "y"
        )
        axes[0][i].figure.set_figwidth(20)

        axes[0][i].xaxis.set_major_locator(AutoLocator())
        axes[0][i].yaxis.set_major_locator(AutoLocator())

        axes[0][i].grid(which="major", alpha=0.5)
        axes[0][i].grid(which="minor", alpha=0.2)

        if len(plot_values[i][0]) < 50:
            axes[0][i].plot(plot_values[i][0], plot_values[i][1],
marker="o", color="black")
        else:
            axes[0][i].plot(plot_values[i][0], plot_values[i][1],
marker=",", color="black")

    fig.savefig(file_name, bbox_inches="tight")
    plt.close(fig)


draw_plots(
        [
            (x_shifted, np.array(list(map(FUNC, x_shifted)))),
            (x_shifted, lagrange_interpolation(x_shifted)),
            (x_shifted, spline_interpolation(x_shifted))
        ],
        [
            "Original function",
            "Lagrange interpolation",
            "Spline interpolation"
        ],
        "plots_shifted.png"
    )
```

Графики функции и её аппроксимаций:

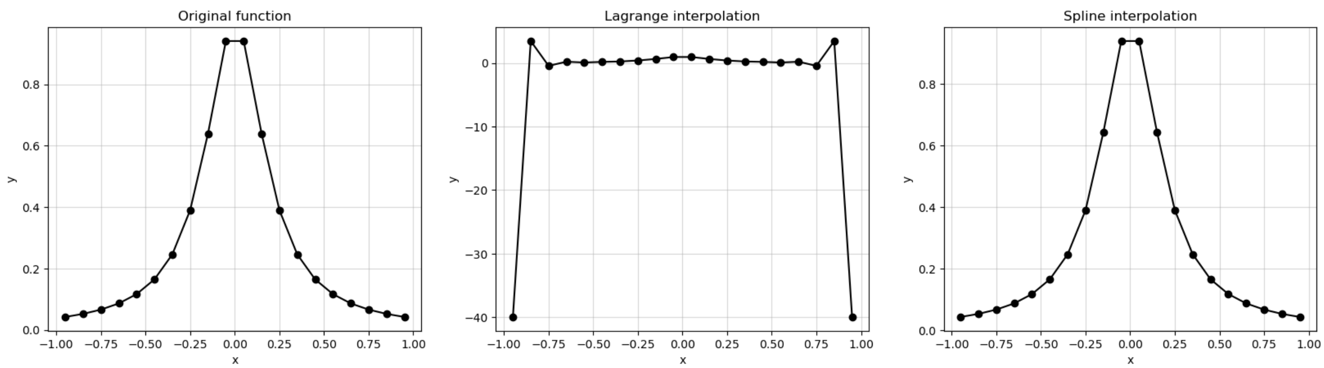

Рис. 1.1 – Графики функции и её аппроксимаций

Построим графики погрешностей. Графики строятся по точкам -1 + 0.5k (k = 0,1…40)

```
y_fine = np.array(list(map(FUNC, x_fine_sample)))
    draw_plots(
        [
            (x_fine_sample, lagrange_interpolation(x_fine_sample) -
y_fine),
            (x_fine_sample, spline_interpolation(x_fine_sample) - y_fine)
        ],
        [
            "Lagrange error",
            "Spine error"
        ],
        "Approximation error"
    )
```

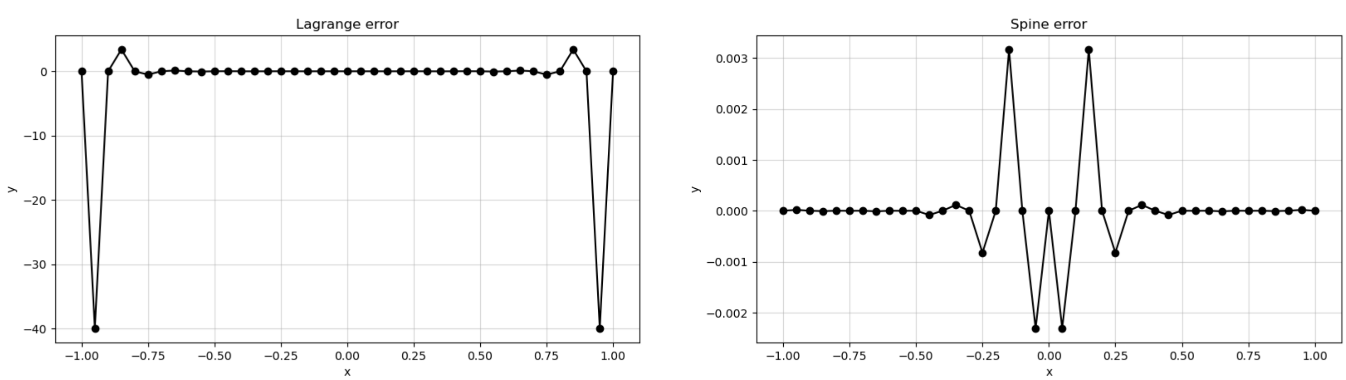

Рис. 1.2 – Погрешность полиномов

Как видно из графиков полиномов, а так же из графиков их погрешности, интерполяционный полином Лагранджа очень плохо аппроксимирует исходную функцию, с погрешностью, превышающей значение функции в несколько сотен раз в нескольких точках. Данный эффект можно объяснить феномоном Рунге: для некоторых функций, интерполянт, полученный полиноминальной интерполяцией, будет осцилировать ближе к концам промежутка, а погрешность интерполяции будет стремится к бесконечности с возростанием степени полинома. Из этого можно сделать вывод, что Полином лагранджа не подходит для аппроксимации данной функции.

Задача вычисления интегралов:

Построим графики подынтегральных функций для m = -1 и m = -0.5:

```
def draw_func():
    func_x = np.arange(START + 0.001, END + 0.01, 0.01)

    fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
    fig.set_figwidth(15)

    ax1.set_ylim([0, 10])
    ax2.set_ylim([0, 10])
    ax1.grid()
    ax2.grid()

    ax1.title.set_text("m = -1")
    ax2.title.set_text("m = -0.5")
    ax1.set(xlabel="x", ylabel="y")
    ax2.set(xlabel="x", ylabel="y")

    ax1.plot(func_x, np.array(list(map(INTEGRAL_FUNC_1, func_x))))
    ax2.plot(func_x, np.array(list(map(INTEGRAL_FUNC_2, func_x))))

    fig.savefig("Integral function", bbox_inches="tight")
    plt.close(fig)
```

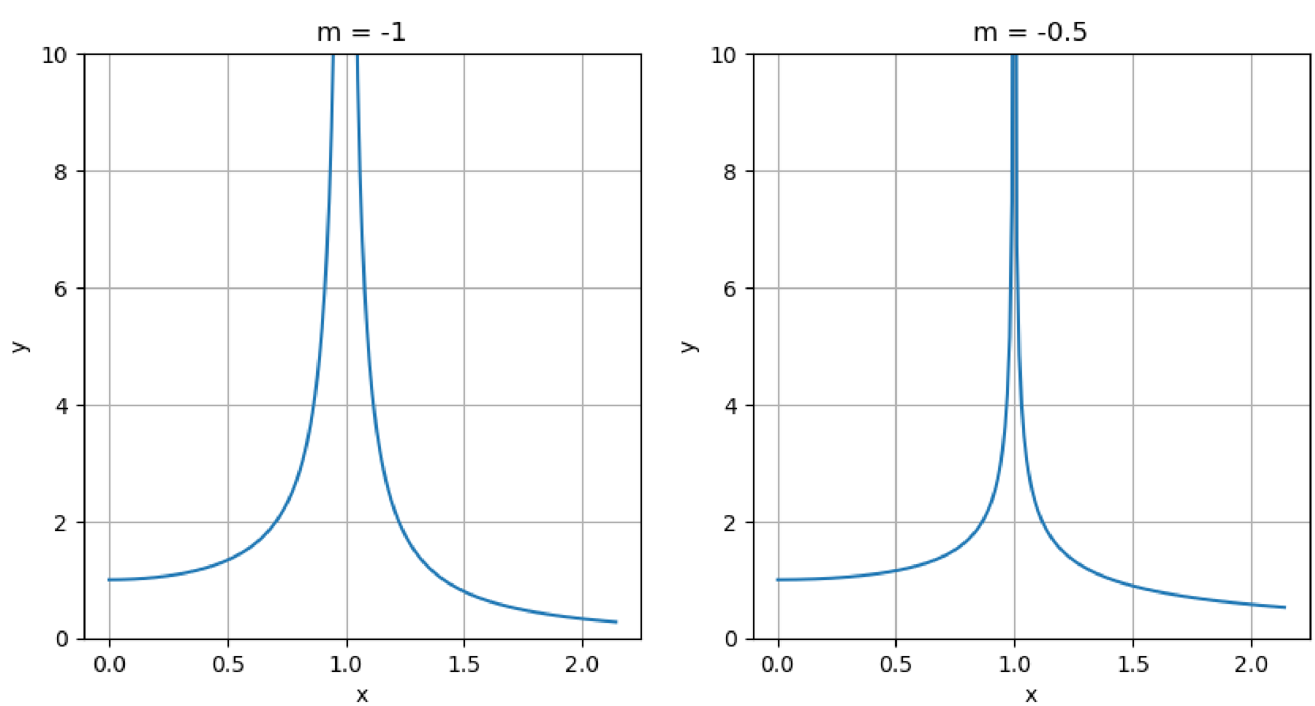

Рис. 2.1 – Графики подинтегральных функций

Как видно из графиков, обе функции терпят разрыв в точке x = 1, эта точка будет отдельно обрабатываться при вычислении интеграла. Для этого функция QUANC8 будет вызываться два раза, первый раз для вычисления интеграла на интервале [0, 1 – eps], второй раз на интеревале [1 + eps, 2.14], где eps > 0.

В решении используется функция из библиотеки SciPy – quad, которая является аналогом QUANC8. В этой функции присутствует ограничение на максимальное количество вычислений подинтегральной функции, зная это, чтобы найти максимально точное приближение значения интеграла, необходимо подобрать такое значение eps, при котором будет достигаться предел количества вычислений подинтегральной функции.

Функция для вычисления интеграла с заданным приближением к точке разрыва:

```
def get_integral_value(func, eps):
    return quad(func, START, 1 - eps, limit=30)[0] +\
           quad(func, 1 + eps, END, limit=30)[0]
```

Вычисление значений интеграла для различных приближений eps:

```
    eps_val = []
    integ_val_1 = []
    integ_val_2 = []

    eps = 0.1
    for i in range(0, 20):
        integ_val_1.append(get_integral_value(INTEGRAL_FUNC_1, eps))
        integ_val_2.append(get_integral_value(INTEGRAL_FUNC_2, eps))

        eps_val.append("1e-" + str(i+1))
        eps = eps / 10
```

Далее выведем полученные значения интегралов в консоль. Определение функции print_table можно найти в предыдущем пункте:

```
print_table(["eps", "integral, m=-1", "integral, m=-0.5"],
                (eps_val, integ_val_1, integ_val_2))
```

Полученная таблица:

eps	integral, m=-1	integral, m=-0.5
1e-1	2.4878834396880563	2.070459596583404
1e-2	4.7917075976349	2.68221141995471
1e-3	7.094305065785264	2.8756119419877395
1e-4	9.39689028252932	2.9367703915120233
1e-5	11.699475376757878	2.956110390854767
1e-6	14.002060469719659	2.962226235640096
1e-7	16.304645563378347	2.9646074445311195
1e-8	18.607230653733133	2.965054662999467
1e-9	20.909815752183484	2.9650546636380235
1e-10	23.212401258332562	2.965054662808263
1e-11	25.514646034790847	2.9650546625902847
1e-12	27.360736106164865	2.965054662670297
1e-13	27.867001205465883	2.965054662660156
1e-14	27.927972949424017	2.96505466273993
1e-15	27.934168557470244	2.9650546627607843
1e-16	27.93492597371344	2.965054662763673
1e-17	27.934967033256825	2.9650546627638423
1e-18	27.934967033256825	2.9650546627638423
1e-19	27.934967033256825	2.9650546627638423
1e-20	27.934967033256825	2.9650546627638423

Как можно видеть из таблицы, максимальное значение вычислений подинтегральной функции достигается при eps = 1e-15 и eps = 1e-8 для первого и второго интеграла соответственно.

Вывод:

В ходе выполнения данной лобараторной работы, я ознакомился с функциями SPLINE, SEVAL и QUANC8. По результатам работы я получил опыт обработки исключительных ситуаций при вычислении значения интегралов с разрывами подынтегральной функции. Полученные результаты из первого пункта работы позволяют сделать вывод, что использование сплайн интерполяции предпочтительнее, чем использование интерполяционных полиномов при интерполяции с большим количеством точек.