ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Доцент		А.В. Аграновский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 2

СПЛАЙНОВАЯ КРИВАЯ БЕЗЬЕ

Вариант 5

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4128		В.А. Воробьев
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2023

СОДЕРЖАНИЕ

1 Цель работы

Изучение сплайновой кривой Безье, построение сплайновой кривой Безье с помощью математического пакета и/или языка программирования высокого уровня.

Задание:

1) Построить график гармонических колебаний.
2) На периоде гармонических колебаний взять N точек, где N равно 4 плюс номер студента в группе.
3) По опорным точкам из пункта 2 построить кривую Безье (на том же графике, что и в пункте 1).
4) Рассчитать ошибку восстановления гармонических колебаний кривой Безье.
5) Уменьшить число точек на периоде в 2 раза и повторить пункты 1-4.
6) Увеличить число точек на периоде в 2 раза и повторить пункты 1-4.
7) Построить кривую Безье на основе полинома N-го порядка (где N берется из пункта 2) и рассчитать ошибку.
8) На форме должно быть 3 кнопки: отображение графика гармонических колебаний, кривой Безье на основе гармонических колебаний и кривой Безье на основе N-го полинома

Группа	Функция для лабораторных работ 1 и 2	Интервал
4128	*f(x) = 2 sin(x) + 1.5 sin(2x)*	*один период f(x)*

Рисунок 1 – Вариант задания

2 Теоретические сведения

Сплайн – кривая, удовлетворяющая некоторым критериям гладкости.

Базовые (опорные) точки – набор точек, на основе которых выполняется построение кривой.

Интерполяция – построение кривой, точно проходящей через набор базовых точек.

Полиномом называется функция вида:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = y$$

Существует большое количество разных вариантов сплайновых кривых, отличающихся своими свойствами.

Кривая Безье является частным случаем многочленов Бернштейна, описанных Сергеем Натановичем Бернштейном в 1912 году.

По заданному массиву точек P0, P1, P2, P3 сплайновая кубическая элементарная кривая Безье (По заданному массиву точек P0, P1, P2, P3 сплайновая кубическая элементарная кривая Безье (рисунок 1) описывается уравнением (3): см. рис. 2) описывается уравнением:

$$R(t) = \Big(\big((1-t)P_0 + 3tP_1\big)(1-t) + 3t^2 P_2\Big)(1-t) + t^3 P_3,$$

$$0 \le t \le 1.$$

Элементарная кривая начинается в точке P0 и заканчивается в точке P3, касаясь при этом отрезков P0 P1 и P2 P3.

Свойства составной кривой Безье:

1) проходит внутри выпуклой оболочки, заданной опорными точками;
2) набор базовых функций однозначно определяет кривую, т.е. нет возможности регулировать ее форму.

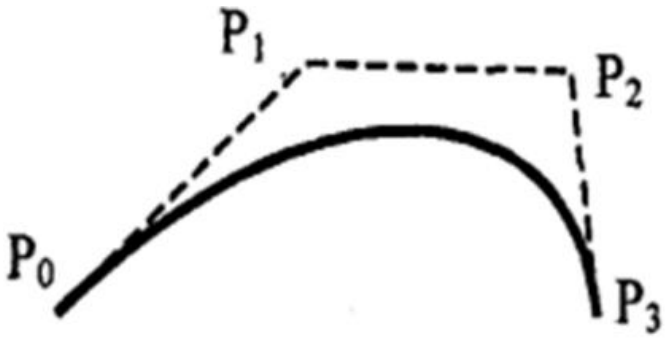

Рисунок 2 – Пример построения кривой Безье

Чтобы составная кривая Безье была геометрически непрерывной, необходимо, чтобы каждые три точки в месте стыковки лежали на одной прямой. Составную кривую построим из наборов элементарных кривых Безье для четверок вершин.

Допустим, у нас есть шесть точек P1, P2, P3, P4, P5, P6, где Pi(xi, yi). Пусть (x3; y3) и (x4; y4) — координаты третьей и четвертой точки соответственно. Вставляем между ними дополнительную точку P’ с координатами x’=(x3+x4)/2 и y’=(y3+y4)/2, после чего проводим одну кривую через точки P1, P2, P3, P’ а вторую — через точки P’, P4, P5, P6. В результате получим одну гладкую кривую для шести точек.

Если точек больше шести, их нужно разбить по такой же схеме и связать полученные группы с помощью дополнительных точек, как описано выше.

Последнюю точку можно повторить несколько раз, если множество точек не делится на целое число групп, чтобы кривая доходила до последней точки.

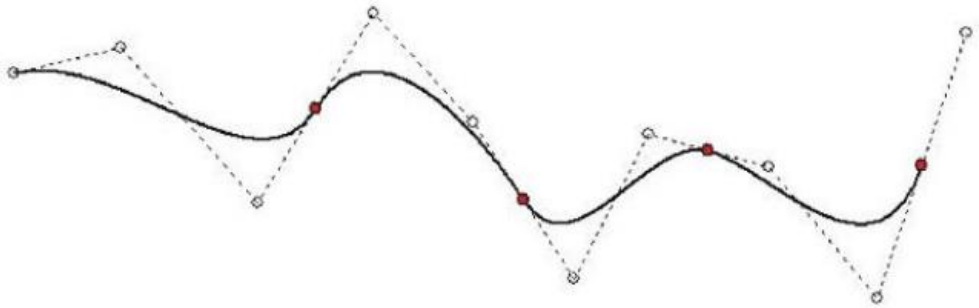

Рисунок 3 – Пример аппроксимации кривой Безье

3 Выполнение работы

Для выполнения лабораторной работы было решено выбрать высокоуровневый язык Python 3 и библиотеки matplotlib (для отображения графиков) и NumPy (работа с матрицами). Код, выполняющий поставленную задачу, показан в приложении А, а также доступен по ссылке на GitHub (URL: https://github.com/vladcto/SUAI_homework/blob/b6f8ae8b479f5aaa19e9cc8c6a41d25f5db6dec2/4_semester/CG/2%D0%BB%D1%80/solution.py). Код снабжен комментариями и легок для понимания. Вся логика вычисления точек кривой Безье на основе опорных точек сосредоточена в функции bezier_curve (см. приложение А).

Алгоритм построения кривой Безье для N точек выглядит так:

1) Берем 3 точки, а в качестве четвертой берем среднюю между 3 и 4 точкою. Строим кривую Безье для 4 точек.

2) Если есть следующие 3 точки, то переходим к п. 3), иначе переходим к п. 4).

3) Берем следующие две точки. В качестве первой точки берем среднюю точку между последней в предыдущем наборе и первой в текущем наборе. Для 2 и 3 точки берем текущие точки в наборе. Четвертая точка для кривой является средней между последней точкой в наборе и первой точкой следующего набора. Строим кривую Безье для 4 точек. Переходим к п. 2).

4) В качестве 1 точки берем среднюю точку между последней в предыдущем наборе и первой в текущем наборе. Затем берем оставшиеся точки и дополняем их копиями последней, чтобы всего стало 4 точки. Строим кривую Безье для 4 точек.

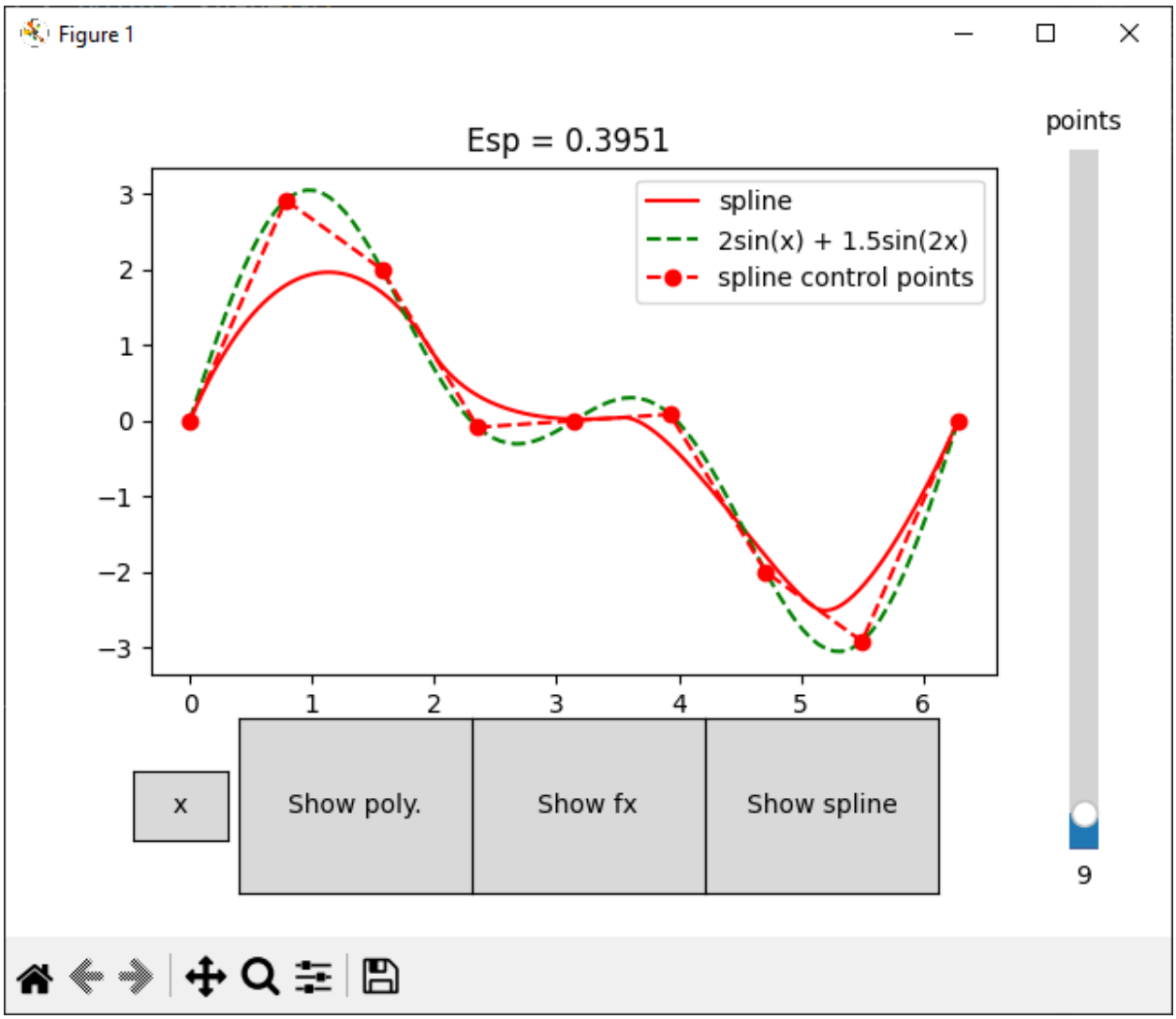

Рисунок 4 – Кривая Безье для 9 контрольных точек

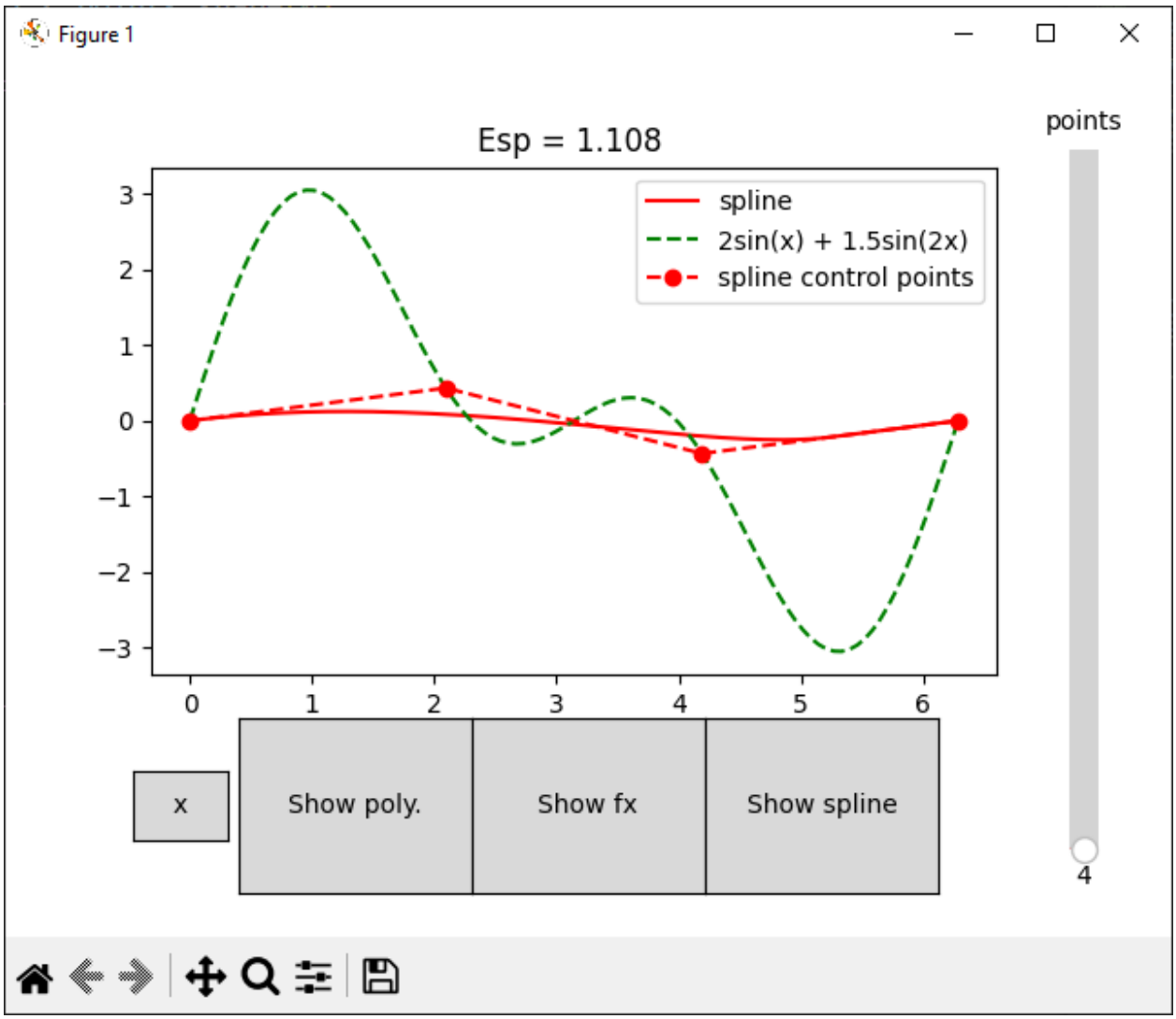

Рисунок 5 – Кривая Безье для 4 точек

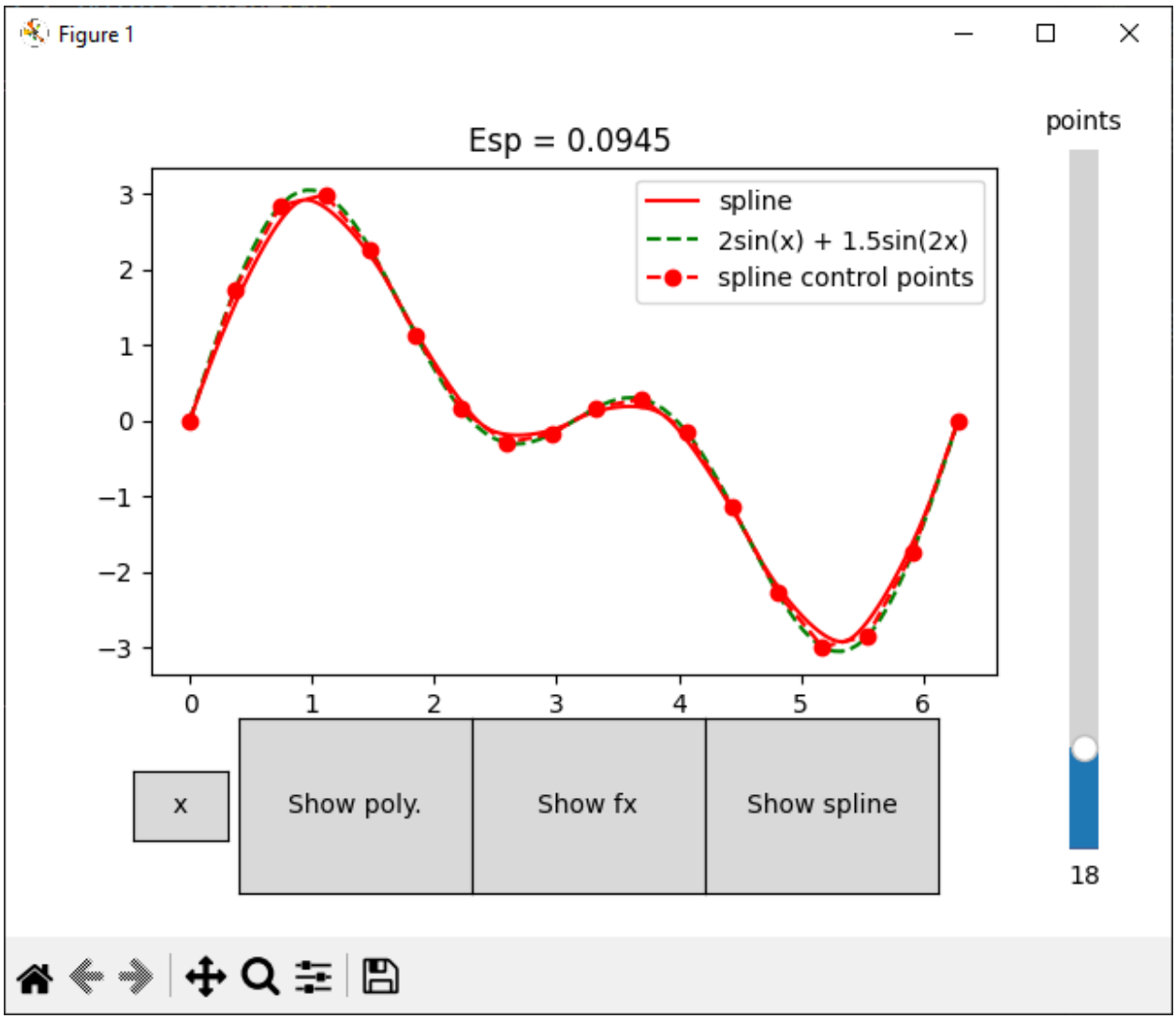

Рисунок 6 – Кривая Безье для 18 точек

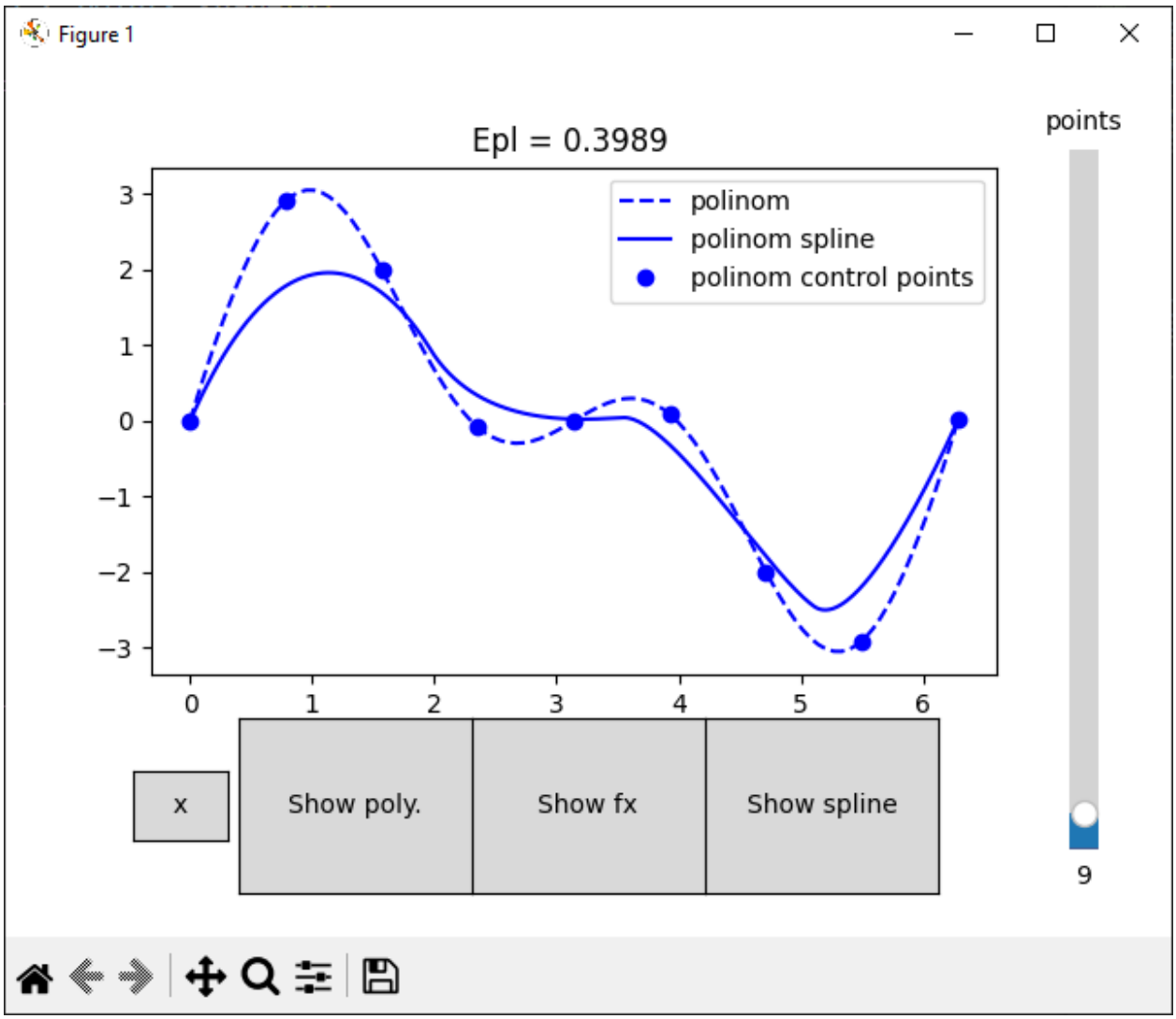

Рисунок 7 – Кривая Безье для полинома 9-го порядка

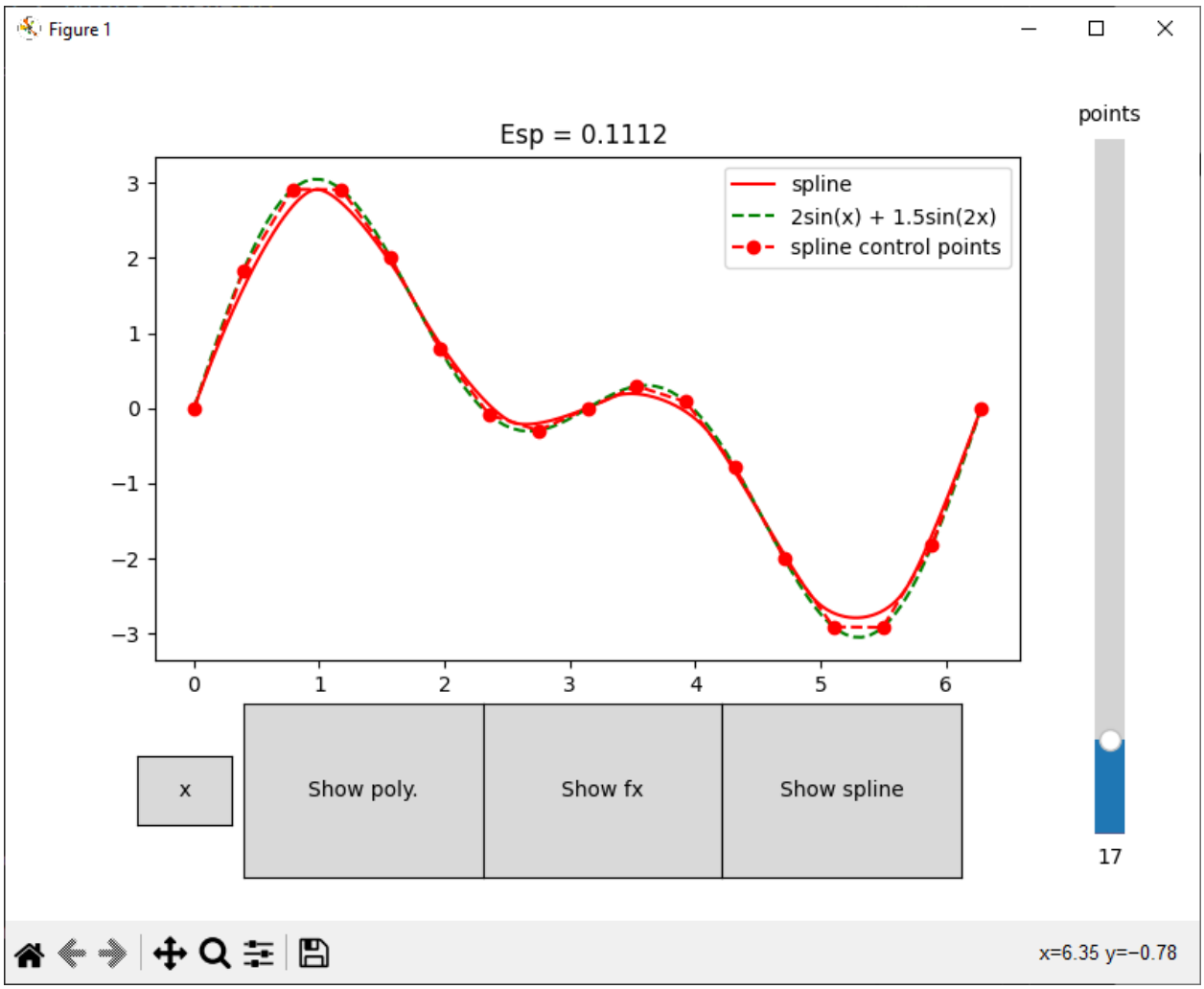

Рисунок 8 – Кривая Безье для произвольного количества точек

4 Вывод

В результате выполнения лабораторной работы были получены навыки построения кривой Безье, используя язык Python. С помощью библиотеки matplotlib и NumPy был реализован алгоритм построения кривой для гармонической функции и полинома 9-го порядка, а также кривой Безье для заданного количества опорных точек.

Функционал программы был протестирован, в результате чего мы получили значения отклонения кривой Безье от графика гармонической функции, для которой был построен сплайн (см. таблица 1):

Таблица 1 – Ошибка восстановления

N точек	Ошибка восстановления
4	1.108
9	0.3951
18	0.0945

По полученным значениям можем утверждать, что с увеличением количества опорных точек увеличивается точность кривой Безье.

Выделим несколько преимуществ кривой Безье:

1) Можно изменять кривизну линии, регулируя точки, не являющиеся концами. Эти точки выступает некого рода весами для отклонения кривой.
2) Сложные фигуры могут быть сделаны из нескольких кривых Безье, что является более простой задачей, чем высчитывание коэффициентов полинома.

В качестве недостатка можно обозначить, что кривая Безье проходит только через начальную и конечную опорную точку, тем самым если условие прохождение опорных точек является обязательным, то кривые Безье не подойдут для решения такого рода задач.

Вышеуказанные особенности являются основообразующими для широкого спектра применения, например в таких сферах как:

1) В компьютерной графике, моделировании, в графических редакторах. Шрифты описываются с помощью кривых Безье.
2) В веб-разработке – для графики на Canvas или в формате SVG.
3) CSS-анимации для задания траектории или скорости передвижения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Половко А.М., Бутусов П.Н. Интерполяция. Методы и компьютерные технологии их реализации. – СПб.: БХВ-Петербург, 2004. – 320 с. (дата обращения: 02.03.2023)
2) NumPy: Документация NumPy: сайт. – URL: https://numpy.org/doc/stable/index.html (дата обращения: 01.03.2023)
3) Учебник JavaScript: Кривые Безье: сайт. – URL: https://learn.javascript.ru/bezier-curve (дата обращения: 03.03.2023)
4) Matplotlib: Документация Matplotlib: сайт. – URL: https://matplotlib.org/stable/index.html (дата обращения: 01.03.2023)
5) StackOverflow: Кривые Безье для N точек: сайт. – URL: https://stackoverflow.com/questions/7715788/find-bezier-control-points-for-curve-passing-through-n-points (дата обращения: 03.03.2023)

ПРИЛОЖЕНИЕ А

ЛИСТИНГ ПРОГРАММЫ

```
from math import sin
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button, Slider



# Bezier function
def bezie_equation(points, t):
    p1 = points[0]
    p2 = points[1]
    p3 = points[2]
    p4 = points[3]
    q1 = ((1 - t) * p1 + 3 * t * p2) * (1 - t) + 3 * (t ** 2) * p3
    return q1 * (1 - t) + (t ** 3) * p4



# Get points for Bezier curve
def bezier_curve(points, step=0.01):
    res = []
    i = 0
    while i < points.shape[0]:
        # Start of curve
```

```
if i == 0:
    points_now = np.vstack(
        [points[i: 3],
         (points[2] + points[3])/2]
    )
    i += 3
# End of curve
elif i + 2 >= points.shape[0]:
    points_now = np.vstack(
        [(points[i] + points[i-1])/2, points[i:],
         *[points[-1] for _ in range(3)]]
    )
    i += 2
# Middle of curve
else:
    points_now = np.vstack(
        [(points[i] + points[i-1])/2,
         points[i: i + 2],
         (points[i+1] + points[i+2])/2]
    )
    i += 2

for t in np.arange(0, 1, step):
    res.append(bezie_equation(points_now, t))
```

```
    return np.array(res)



def update_plot(state, fig):
    fig.clear()
    # Control points for spline and other curves.
    x_points = np.linspace(0, 2 * np.pi, 100)
    x_controls = np.linspace(0, 2 * np.pi, state["points"])
    y_controls = 2 * np.sin(x_controls) + 1.5 * np.sin(x_controls * 2)
    label_text = ""

    if (state["poly"]):
        # Polinom points
        # Polynom points
        y_f = 2 * np.sin(x_points) + 1.5 * np.sin(x_points * 2)
        # Polynom coefs
        p = np.polyfit(x_points, y_f, 9)
        poli_y = np.polyval(p, x_points)
        poli_controls_y = np.polyval(p, x_controls)
        poli_points = bezier_curve(
            np.column_stack([x_controls, poli_controls_y]))

        fig.plot(x_points, poli_y, "b--", label="polinom")
        fig.plot(poli_points[:, 0], poli_points[:, 1],
```

```
            "b-", label="polinom spline")
    fig.plot(x_controls, poli_controls_y, "bo",
            label="polinom control points")
    # Calculate error
    y_predict = np.polyval(p, poli_points[:, 0])
    error_dif = np.abs(poli_points[:, 1] - y_predict).mean()
    label_text += f"Epl = {round(error_dif,4)} "

if (state["bezier"]):
    # Spline points
    splain = bezier_curve(np.column_stack((x_controls, y_controls)))
    x_splain = splain[:, 0]
    y_splain = splain[:, 1]

    fig.plot(x_splain, y_splain, "r-", label="spline")
    # Calculate error
    y_predict = np.asarray(
        [2 * sin(x) + 1.5*sin(2 * x) for x in x_splain])
    error_dif = np.abs(y_splain - y_predict).mean()
    label_text += f"Esp = {round(error_dif,4)} "

if (state["fx"]):
    y_f = 2 * np.sin(x_points) + 1.5 * np.sin(x_points * 2)
```

```
        fig.plot(x_points, y_f, "g--", label="2sin(x) + 1.5sin(2x)")
        fig.plot(x_controls, y_controls, "ro--", label="spline control points")

    handles, labels = fig.get_legend_handles_labels()
    by_label = dict(zip(labels, handles))
    fig.legend(by_label.values(), by_label.keys())
    fig.set_title(label_text)
    plt.draw()


def draw_polinom(state, fig):
    state["poly"] = True
    update_plot(state, fig)


def draw_spline(state, fig):
    state["bezier"] = True
    update_plot(state, fig)


def draw_function(state, fig):
    state["fx"] = True
    update_plot(state, fig)
```

```
def clear(state, fig):
    state["fx"] = False
    state["bezier"] = False
    state["poly"] = False
    update_plot(state, fig)


def change_control_points(state, fig, num):
    state["points"] = int(num)
    update_plot(state, fig)


# Draw graphics
fig, ax = plt.subplots()
my_state = {"fx": False, "bezier": False, "poly": False, "points": 4}
fig.subplots_adjust(bottom=0.3)
fig.subplots_adjust(right=0.85)

# Buttons and slider
ax_poli_btn = fig.add_axes([0.2, 0.05, 0.2, 0.2])
poli_btn = Button(ax_poli_btn, "Show poly.")
poli_btn.on_clicked(lambda _: draw_polinom(my_state, ax))
```

```
ax_fx_btn = fig.add_axes([0.4, 0.05, 0.2, 0.2])
fx_btn = Button(ax_fx_btn, "Show fx")
fx_btn.on_clicked(lambda _: draw_function(my_state, ax))

ax_splain_btn = fig.add_axes([0.6, 0.05, 0.2, 0.2])
splain_btn = Button(ax_splain_btn, "Show spline")
splain_btn.on_clicked(lambda _: draw_spline(my_state, ax))

ax_clear_btn = fig.add_axes([0.11, 0.11, 0.08, 0.08])
clear_btn = Button(ax_clear_btn, "x")
clear_btn.on_clicked(lambda _: clear(my_state, ax))

ax_slider = fig.add_axes([0.9, 0.1, 0.05, 0.8])
slider = Slider(ax_slider,
                valmin=4,
                valmax=100,
                orientation="vertical",
                label="points",
                valstep=1)
slider.on_changed(lambda num: change_control_points(my_state, ax, num))
plt.show()
```