ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

| канд. техн. наук, доцент | | А. В. Аграновский |
|---|---|---|
| должность, уч. степень, звание | подпись, дата | инициалы, фамилия |

# ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 6

# СПЛАЙНОВАЯ КРИВАЯ БЕЗЬЕ

по курсу:

КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

| СТУДЕНТ гр. № | 4326 | | Г. С. Томчук |
|---|---|---|---|
| | | подпись, дата | инициалы, фамилия |

Санкт-Петербург 2024

## 1 Цель работы

Цель работы — освоение методов построения сплайновой кривой Безье, а также реализация её построения с использованием математического программного обеспечения или языка программирования высокого уровня.

## 2 Формулировка задания

Работа выполнялась по варианту № 20. Задачи состояли в следующем:

1. Построить график функции $f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$ на заданном интервале $x \in [-1; 3]$.
2. На основе графика выбрать N=24 опорных точек.
3. Реализовать построение сплайновой кривой Безье по данным опорным точкам.
4. Рассчитать и отобразить ошибку восстановления функции f(x) с помощью сплайновой кривой Безье.
5. Уменьшить число опорных точек до N=12 и повторить пункты 3–4.
6. Увеличить число опорных точек до N=48 и повторить пункты 3–4.
7. Построить кривую Безье на основе полинома N-го порядка, где N равно количеству опорных точек, и рассчитать ошибку восстановления функции для каждого из случаев.

## 3 Теоретические сведения

1. Определение кривой Безье

Кривая Безье – это параметрическая кривая, задаваемая на основе набора опорных точек $P_0, P_1, \dots, P_n$. Координаты точки кривой рассчитываются по формуле:

$$\mathbf{B}(t) = \sum_{i=0}^{n} B_{i,n}(t) \cdot \mathbf{P}_i, \quad t \in [0,1],$$

где:

- $\mathbf{B}(t)$ — точка на кривой,
- $\mathbf{P}_i = (x_i, y_i)$ — координаты опорных точек,
- $B_{i,n}(t)$ — полиномы Бернштейна.

Полиномы Бернштейна определяются как:

$$B_{i,n}(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \quad i = 0, 1, \dots, n,$$

где:

$$\binom{n}{i} = \frac{n!}{i! \cdot (n-i)!}$$

— биномиальный коэффициент.

3. Ошибка восстановления

Ошибка восстановления функции f(x) с помощью кривой Безье рассчитывается как среднеквадратическое отклонение:

$$E = \sqrt{\frac{1}{m} \sum_{j=1}^{m} (f(x_j) - f_{\text{Bezier}}(x_j))^2},$$

где:

- $x_j$ — точки интервала $x \in [-1, 3]$,
- $f(x)$ — значение исходной функции,
- $f_{\text{Bezier}}(x_j)$ — значение кривой Безье в этих точках,
- $m$ — количество точек на интервале.

4. Полином Безье

Полином Безье N-го порядка строится на основе опорных точек, и его координаты вычисляются как интерполяция функции через полиномы:

$$x(t) = \sum_{i=0}^{n} c_i \cdot t^i, \quad y(t) = \sum_{i=0}^{n} d_i \cdot t^i,$$

где $c_i$ и $d_i$ — коэффициенты полиномов, получаемые из решения системы линейных уравнений, аппроксимирующих опорные точки.

## 4 Скриншоты, иллюстрирующие результаты работы программы

На рис. 1, 2 и 3 представлены скриншоты окна программы, демонстрирующие соответственно кривую исходной функции, кривую Безье и полиноминальную кривую Безье.

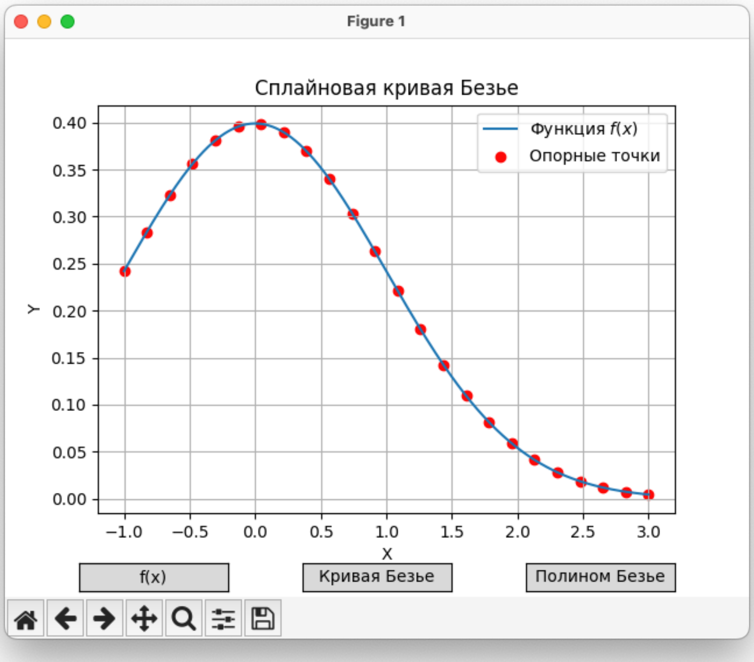


Рисунок 1

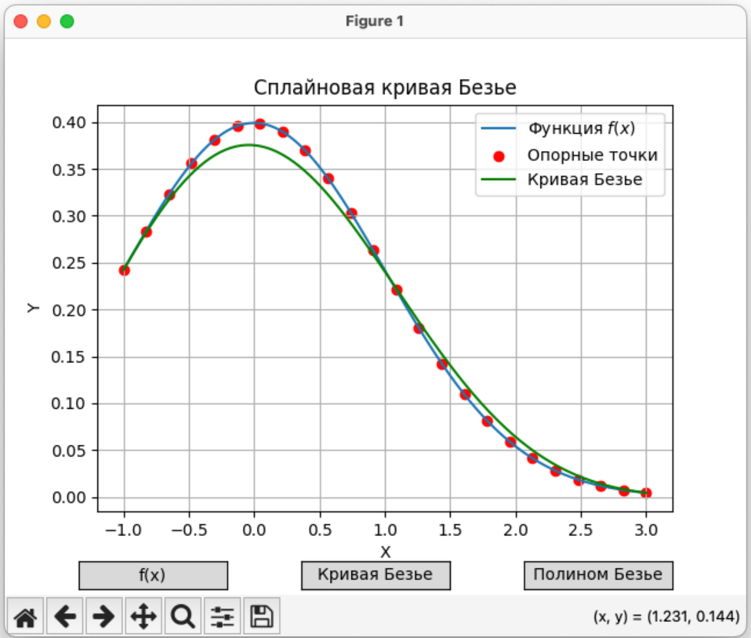


Рисунок 2

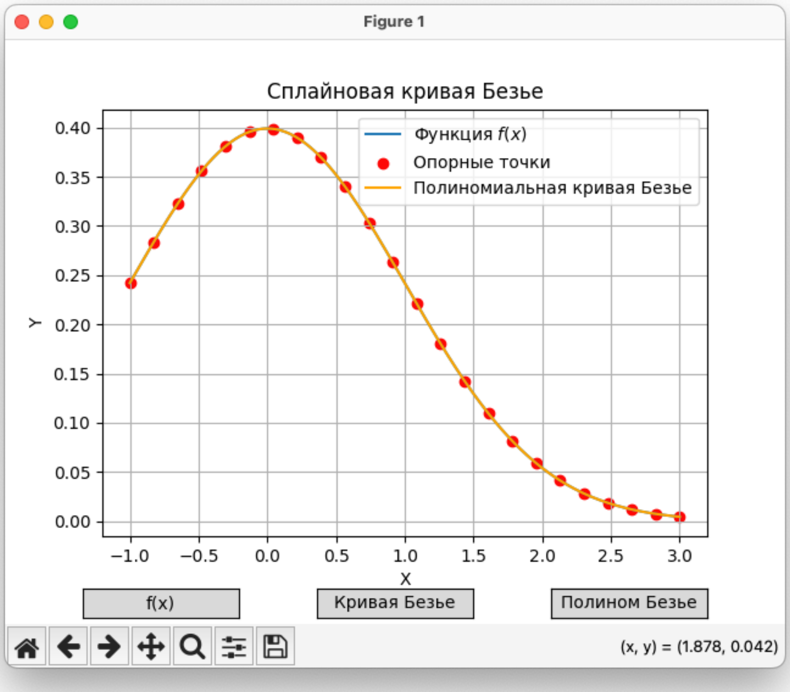


Рисунок 3

На рис. 4 изображен вывод из консоли Python, в котором содержатся вычисленные значения ошибок восстановления кривой Безье.

```
Ошибка восстановления (24 точки): 0.012682
Ошибка восстановления для полинома: 0.000008
```

Рисунок 4

# 5 Вывод

1. В ходе выполнения лабораторной работы были изучены теоретические основы сплайновой кривой Безье, ее построение с использованием полиномов Бернштейна, а также применение для аппроксимации функций.
2. Реализованы алгоритмы построения кривой Безье и полиномиальной кривой Безье на основе 24 опорных точек, равномерно распределенных на интервале $x \in [-1,3]$. Для этого была использована стандартная функция $f(x)$, представляющая нормальное распределение.
3. Проведен анализ точности восстановления исходной функции:
   - Ошибка восстановления функции при построении стандартной кривой Безье составила 0.012682. Это объясняется ограничением точности кривой Безье из-за локальности влияния опорных точек и применения полиномов Бернштейна.
   - При построении полиномиальной кривой Безье ошибка восстановления значительно снизилась до 0.000008, что свидетельствует о высокой точности интерполяции на основе полиномов высокой степени.
4. Анализ ошибок показал, что стандартная кривая Безье подходит для приближенной аппроксимации функций, но для задач с высокой точностью предпочтительно использовать полиномиальные аппроксимации. Однако, при увеличении числа опорных точек или порядка полиномов, возможно появление эффекта "Рунге" (осцилляций), что требует дополнительного анализа устойчивости решения.
5. Полученные результаты демонстрируют эффективность математических инструментов для решения задач аппроксимации функций, а также важность выбора подходящей методики в зависимости от требуемой точности и свойств исходной функции.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. - СПб.: БХВ-Петербург, 2003. – 560 с.
2. Петров, А. Н., Смирнов, И. В. Основы программирования на Python: Учебное пособие. – Санкт-Петербург: Издательство СПб ГУАП, 2022. – 320 с.
3. Бобылев, С. И. Основы компьютерной графики: учебное пособие / С. И. Бобылев. — М.: Издательство МГТУ им. Н. Э. Баумана, 2010. — 224 с.
4. Ласков, А. В. Математические основы компьютерной графики / А. В. Ласков, Ю. Г. Лоскутов. — СПб.: Питер, 2008. — 352 с.
5. NumPy. Documentation. URL: https://numpy.org/doc/stable/ (дата обращения: 27.10.2024).
6. Matplotlib. Documentation. URL: https://matplotlib.org/stable/contents.html (дата обращения: 27.10.2024).

# ПРИЛОЖЕНИЕ А

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Button
from scipy.special import comb
from numpy.polynomial.polynomial import Polynomial


# Исходная функция
def f(x):
    return (1 / np.sqrt(2 * np.pi)) * np.exp(-x**2 / 2)


# Функция для вычисления стандартной кривой Безье
def bezier_curve(control_points, num_points=200):
    length = len(control_points) - 1  # Порядок полинома
    t = np.linspace(0, 1, num_points)  # Параметризация
    curve = np.zeros((num_points, 2))  # Координаты кривой
    for i, (x, y) in enumerate(control_points):
        bernstein_poly = comb(length, i) * (t ** i) * ((1 - t) **
(length - i))
        curve[:, 0] += bernstein_poly * x
        curve[:, 1] += bernstein_poly * y
    return curve[:, 0], curve[:, 1]


# Функция для построения полиномиальной кривой Безье
def polynomial_bezier_curve(control_points, num_points=200):
    x_points, y_points = control_points[:, 0], control_points[:, 1]

    # Строим полиномы для x(t) и y(t) через индексацию
    poly_x = Polynomial.fit(np.linspace(0, 1, len(x_points)),
x_points, len(x_points) - 1)
    poly_y = Polynomial.fit(np.linspace(0, 1, len(y_points)),
y_points, len(y_points) - 1)

    # Вычисляем значения кривой
    t = np.linspace(0, 1, num_points)
    curve_x = poly_x(t)
    curve_y = poly_y(t)

    return curve_x, curve_y


# Функция для вычисления ошибки восстановления
def calculate_error(original_y, interpolated_y):
    return np.sqrt(np.mean((original_y - interpolated_y) ** 2))


# Основной класс приложения
class BezierLab:
    def __init__(self, ax):
```

```
        self.ax = ax
        self.ax.set_title("Сплайновая кривая Безье")
        self.ax.set_xlabel("X")
        self.ax.set_ylabel("Y")
        self.ax.grid(True)

        # Исходные данные
        self.x_range = np.linspace(-1, 3, 1000)
        self.y_original = f(self.x_range)
        self.control_points = None

        # Инициализация кнопок
        ax_button1 = plt.axes([0.1, 0.01, 0.2, 0.05])
        ax_button2 = plt.axes([0.4, 0.01, 0.2, 0.05])
        ax_button3 = plt.axes([0.7, 0.01, 0.2, 0.05])
        self.btn1 = Button(ax_button1, 'f(x)')
        self.btn2 = Button(ax_button2, 'Кривая Безье')
        self.btn3 = Button(ax_button3, 'Полином Безье')

        self.btn1.on_clicked(self.plot_func)
        self.btn2.on_clicked(self.plot_bezier)
        self.btn3.on_clicked(self.plot_bezier_polynomial)

        self.reset_plot()

    def reset_plot(self):
        self.ax.clear()
        self.ax.grid(True)
        self.ax.set_title("Сплайновая кривая Безье")
        self.ax.set_xlabel("X")
        self.ax.set_ylabel("Y")

    def plot_func(self, event=None):
        self.reset_plot()
        self.ax.plot(self.x_range, self.y_original, label='Функция
$f(x)$')
        self.control_points = self.get_control_points(24)
        self.ax.scatter(self.control_points[:, 0],
self.control_points[:, 1], color='red', label='Опорные точки')
        self.ax.legend()
        plt.draw()

    def plot_bezier(self, event=None):
        self.reset_plot()
        self.plot_func()

        bezier_x, bezier_y = bezier_curve(self.control_points)
        self.ax.plot(bezier_x, bezier_y, label='Кривая Безье',
color='green')
        self.ax.legend()

        error = calculate_error(self.y_original,
np.interp(self.x_range, bezier_x, bezier_y))
```

```
        print(f"Ошибка восстановления (24 точки): {error:.6f}")
        plt.draw()

    def plot_bezier_polynomial(self, event=None):
        self.reset_plot()
        self.plot_func()

        # Кривая Безье на основе полинома 24-го порядка
        bezier_x, bezier_y = 
polynomial_bezier_curve(self.control_points)
        self.ax.plot(bezier_x, bezier_y, label='Полиномиальная кривая 
Безье', color='orange')
        self.ax.legend()

        error = calculate_error(self.y_original, 
np.interp(self.x_range, bezier_x, bezier_y))
        print(f"Ошибка восстановления для полинома: {error:.6f}")
        plt.draw()

    def get_control_points(self, n_points):
        x_points = np.linspace(-1, 3, n_points)
        y_points = f(x_points)
        return np.column_stack((x_points, y_points))


# Основной блок запуска программы
fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.15)
app = BezierLab(ax)
plt.show()
```