Отчет по лабораторной работе: интерполяция и сплайн для контура автомобиля

Артем Балакирев

Содержание

1 Введение

Цель данной лабораторной работы – исследование методов интерполяции и сплайн-интерполяции на основе оцифрованных данных контура автомобиля. В ходе выполнения были использованы два основных метода: интерполяционный полином Лагранжа и кубический сплайн с краевыми условиями типа (ii) на левом конце и типа (i) на правом.

2 Теоретические основы

2.1 Интерполяционный полином Лагранжа

Интерполяционный полином Лагранжа – это способ построить функцию, которая проходит через конкретные заданные точки. У нас есть несколько точек, например (x_0, y_0), (x_1, y_1) и так далее, и мы хотим нарисовать плавную линию, которая пройдет точно через каждую из них. Полином Лагранжа позволяет это сделать, создавая формулу для функции, которая будет идеально «попадать» в каждую точку.

Полином Лагранжа для набора точек $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$ имеет вид:

$$L(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \tag{1}$$

где $L(x)$ – это значение полинома для любого x; n – количество точек минус 1; x_i и y_i – координаты заданных точек; дробь $\frac{x-x_j}{x_i-x_j}$ называется «множителем Лагранжа».

Как работает эта формула? В этом выражении:

- Сумма $\sum_{i=0}^{n}$ означает, что мы складываем несколько выражений, каждое из которых рассчитано для конкретной точки (x_i, y_i).

- Произведение $\prod_{\substack{j=0 \\ j\neq i}}^{n}$ означает, что мы перемножаем несколько дробей, где j изменяется от 0 до n, пропуская значение $j = i$.

Каждое слагаемое внутри суммы $\sum_{i=0}^{n}$ является вкладом конкретной точки (x_i, y_i) в общее значение функции $L(x)$. Например, для $i = 0$, множитель $\prod_{\substack{j=0 \\ j\neq 0}}^{n} \frac{x-x_j}{x_0-x_j}$ делает так, чтобы полином Лагранжа равнялся y_0 при $x = x_0$, а для всех других значений x_j он превращался в ноль, исключая их вклад. Это значит, что каждый элемент полинома ответственен за попадание в свою точку, и в сумме они обеспечивают прохождение кривой через все точки.

2.2 Кубический сплайн

Кубический сплайн – это способ построить «гладкую» кривую, которая также проходит через заданные точки. В отличие от полинома Лагранжа, который может иметь сложную форму для большого количества точек, сплайн разбивает весь диапазон значений на небольшие кусочки и в каждом кусочке строит простую функцию – кубический полином.

Для каждого интервала между двумя соседними точками x_i и x_{i+1} сплайн $S_i(x)$ задается следующим образом:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \tag{2}$$

где a_i, b_i, c_i, и d_i – это специальные коэффициенты для данного интервала $[x_i, x_{i+1}]$.

Почему это работает? В отличие от простого полинома, каждый кусок сплайна между точками представляет собой кубический полином, который делает кривую более плавной, поскольку её форма плавно изменяется из одного интервала в другой. Сплайн не только проходит через все точки, но и его наклон и кривизна (первые и вторые производные) также плавно переходят между кусками, что делает его удобным для создания гладких кривых.

Коэффициенты a_i, b_i, c_i и d_i для каждого интервала рассчитываются так, чтобы сплайн был гладким в местах соединений. Это означает, что если мы посмотрим на два соседних кубических полинома $S_{i-1}(x)$ и $S_i(x)$, то их значения и производные совпадают в точке x_i. Таким образом, весь сплайн выглядит как одно целое, несмотря на то, что он состоит из множества отдельных кубических полиномов.

3 Краевые условия

Чтобы сплайн был гладким не только между точками, но и на концах, необходимо задать специальные условия, называемые краевыми. В этом примере используются следующие краевые условия:

- **Левый конец (ii):** Мы задаем первую производную $\frac{dy}{dx}$ (наклон) на левом конце, предполагая, что она равна 0, что означает горизонтальный наклон на краю слева.
- **Правый конец (i):** На правом конце значение функции совпадает с фактическим значением y в последней точке, что говорит о плавном завершении кривой.

Эти условия помогают контролировать поведение сплайна на границах, предотвращая резкие изгибы и делая кривую ещё более плавной.

4 Методы и код

Далее представлен код Python для расчета интерполяционного полинома Лагранжа и кубического сплайна. Каждая строка кода сопоставлена с соответствующими формулами, описанными в теоретических основах.

4.1 Исходные данные

Для оцифровки данных контура автомобиля были получены координаты точек (x, y), которые будут использоваться в вычислениях.

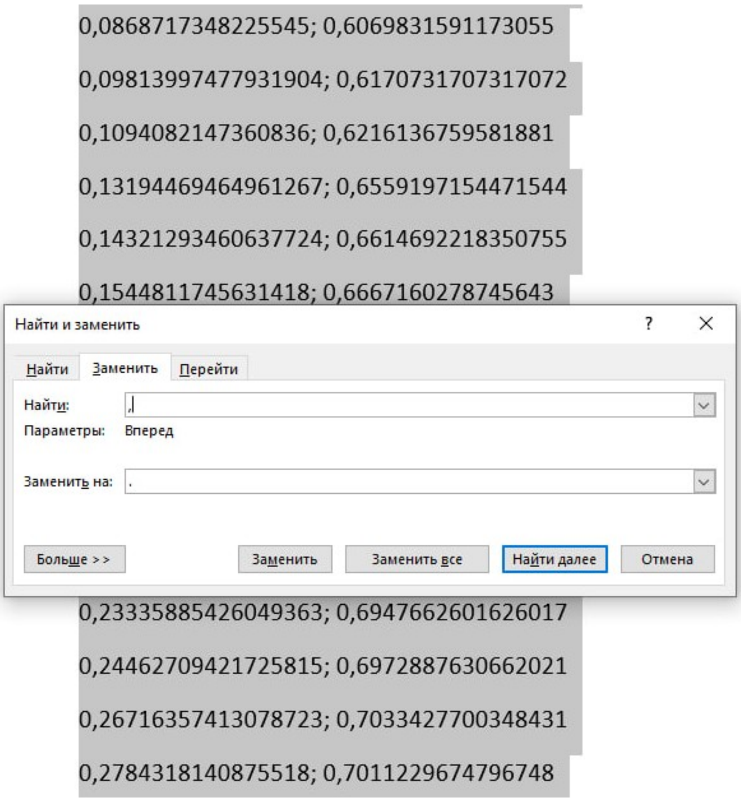

Рис. 1: Форматирование датасета из координат точек

4.2 Код интерполяции Лагранжа (кусочно-линейная интерполяция)

Данный раздел посвящен коду интерполяции на основе полинома Лагранжа с применением метода кусочно-линейной интерполяции. Этот подход делит диапазон точек на интервалы и вычисляет интерполяционный полином для каждого интервала отдельно, что позволяет избежать численных проблем при большом количестве узловых точек и повысить точность вычислений.

```
import numpy as np

def lagrange(x, xs, ys):
s = 0
n = len(xs)
for i in range(n):
p = 1
for j in range(n):
if i == j:
continue
diff = (x - xs[j]) / (xs[i] - xs[j])
if abs(diff) > 1e10:
print(f"Пропуск множителя из-за переполнения при x = {x}")
p = np.nan
break
p *= diff
if not np.isnan(p):
s += ys[i] * p
return s

def lagrange_piecewise(x_intervals, xs, ys):
result_x = []
result_y = []

for interval in x_intervals:
subset_indices = (xs >= interval[0]) & (xs <= interval[1])
xs_subset = xs[subset_indices]
ys_subset = ys[subset_indices]

x_interp = np.linspace(interval[0], interval[1], 50)
y_interp = []

for xi in x_interp:
y_val = lagrange(xi, xs_subset, ys_subset)
if not np.isnan(y_val):
result_x.append(xi)
result_y.append(y_val)
else:
print(f"Пропущено значение NaN для x = {xi}")

return np.array(result_x), np.array(result_y)
```

Листинг 1: Интерполяция Лагранжа (кусочно)

Теоретическое обоснование формулы Лагранжа. Интерполяционный полином Лагранжа позволяет восстановить полином $L(x)$, который проходит через все заданные точки (x_i, y_i) для $i = 0, 1, \ldots, n$. Формула Лагранжа для $n+1$ узловых точек выглядит следующим образом:

$$L(x) = \sum_{i=0}^{n} y_i \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \tag{3}$$

Каждое слагаемое в сумме описывает вклад точки (x_i, y_i) в общее значение $L(x)$ на основе величины y_i, умноженного на выражение, которое учитывает разницу $(x - x_j)$ и нормализует её на $(x_i - x_j)$ для всех $j \neq i$.
Обозначение **произведения** $\prod$ указывает на перемножение всех множителей, и в данном случае оно исключает точку с $j = i$, чтобы избежать деления на ноль. Этот процесс позволяет полиному точно проходить через каждую узловую точку, так как для $x = x_i$ значение $L(x_i) = y_i$.

Разбор формулы Лагранжа в исходном коде. Код реализует эту формулу поэтапно, используя несколько ключевых переменных и проверок:

- **Инициализация переменной результата**. На начальном этапе создается переменная `s = 0`, которая будет аккумулировать итоговое значение полинома $L(x)$.
- **Цикл по узловым точкам**. Внешний цикл `for i in range(n)` отвечает за выполнение расчета для каждой точки i. Здесь n представляет собой количество точек, и каждая итерация по i добавляет вклад от y_i в итоговый полином.
- **Внутренний цикл и расчет произведения**. Внутренний цикл `for j in range(n)` вычисляет произведение $\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$:
  - Условие `if i == j: continue` пропускает итерацию, когда $i = j$, чтобы избежать деления на ноль, что обеспечивает корректность расчета.
  - Каждое выражение $(x - x_j)/(x_i - x_j)$ сохраняется в переменной `diff`.
  - В случае, если `diff` становится слишком большим (например, при $|\texttt{diff}| > 10^{10}$), возникает риск переполнения, поэтому функция выводит предупреждение и присваивает `p` значение `NaN`, предотвращая дальнейшее накопление ошибок.
  - Переменная `p` аккумулирует результат произведения на каждом шаге, постепенно формируя итоговый множитель для i-й точки.
- **Добавление вклада точки в итоговый результат**. После вычисления произведения для точки i, если p не является `NaN`, результат умножается на y_i и добавляется в сумму `s`, обеспечивая тем самым формирование итогового значения $L(x)$.
- **Возврат результата**. Функция возвращает значение полинома $L(x)$ для заданного x, что завершает вычисление интерполяционного полинома Лагранжа для конкретной точки x.

Выбор значений для интерполяции. В процессе интерполяции необходимо определять точки, для которых производится интерполяция.
Параметр `x_interp = np.linspace(interval[0], interval[1], 50)` создает 50 точек внутри каждого интервала, равномерно распределенных от `interval[0]` до `interval[1]`.
Значение 50 выбрано для обеспечения точности и сглаженности кривой, но его можно изменить в зависимости от требований к точности и скорости вычислений. Чем больше точек, тем плавнее будет кривая, но это также увеличивает вычислительную нагрузку.

Описание функции `lagrange_piecewise`. Функция `lagrange_piecewise` предназначена для применения полинома Лагранжа к каждому подмножеству интервалов. Это помогает избежать численных ошибок, которые часто возникают при использовании полиномов высокой степени для большого набора данных.

- **Инициализация переменных для хранения результатов**. Переменные `result_x` и `result_y` аккумулируют значения x и y, интерполированные на каждом интервале, чтобы затем отобразить результат на графике.
- **Цикл по интервалам**. Для каждого интервала из `x_intervals` выбирается подмножество точек из `xs` и `ys`, попадающих в данный интервал. Это реализуется с помощью условия `subset_indices = (xs >= interval[0]) & (xs <= interval[1])`, которое выделяет все точки внутри текущего интервала.
- **Генерация точек интерполяции внутри интервала**. Для каждого выделенного подмножества точек внутри интервала создается равномерная сетка `x_interp`, которая включает 50 точек. Эти точки затем используются для вычисления интерполяционного значения y для каждого x.
- **Вычисление значения интерполяции для каждой точки**. Для каждой точки `xi` из `x_interp` функция `lagrange` вычисляет значение интерполяционного полинома. Если результат не является `NaN`, он добавляется в `result_x` и `result_y`; иначе выводится предупреждение, что значение пропущено.
- **Возврат интерполированных данных**. Функция возвращает массивы `result_x` и `result_y`, содержащие интерполированные значения для всех интервалов, что позволяет строить непрерывную кривую для всего диапазона данных.

Преимущества кусочно-линейного подхода. Использование кусочно-линейного метода для интерполяции снижает численные ошибки, связанные с высокой степенью полинома. Разделение на интервалы делает интерполяцию более устойчивой к выбросам и предотвращает эффекты полиномов высокой степени, такие как эффект Рунге, который проявляется при использовании одного полинома Лагранжа на большом диапазоне значений. Такой подход также позволяет более точно моделировать данные, особенно в случаях, когда они содержат резкие изменения или нелинейные участки.

Визуализация результатов. После вычислений происходит визуализация, показывающая исходные точки и интерполированные значения по кусочно-линейному полиному Лагранжа.

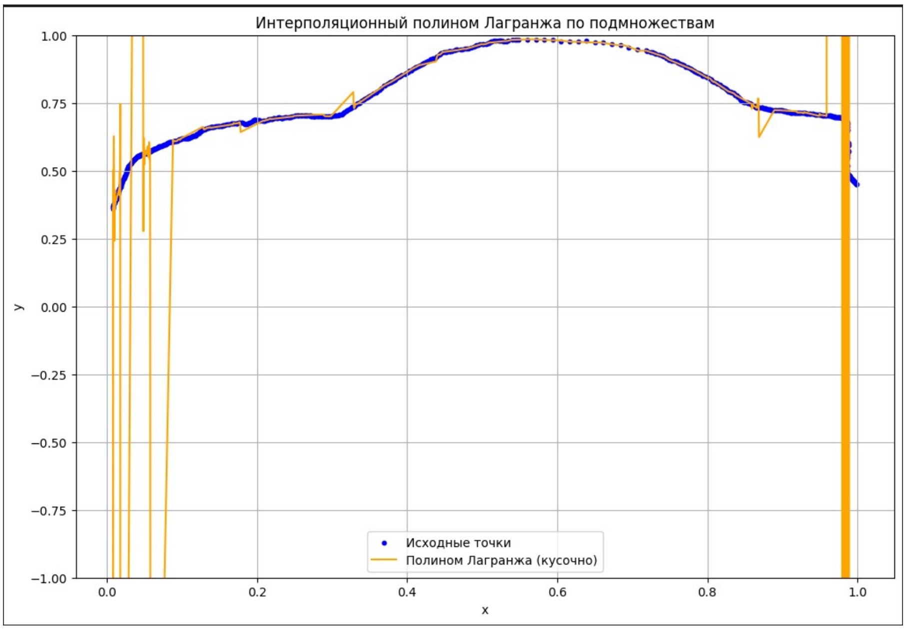

Рис. 2: Полином Лагранжа для данных контура автомобиля

На Рисунке 1 представлен график полинома Лагранжа, который проходит через все исходные точки данных, представляя их с помощью одного многочлена. Однако, поскольку полином Лагранжа имеет глобальную природу, он может показывать значительные отклонения между точками при большом количестве данных.

```
        num_intervals = 30
        x_intervals = np.array_split(np.linspace(min(unique_x), max(unique_x), 100), num_intervals)

        valid_x, valid_y = lagrange_piecewise(x_intervals, unique_x, unique_y)

        plt.figure(figsize=(12, 8))
        plt.scatter(unique_x, unique_y, color='blue', label='Исходнүе точки', s=10)
        plt.plot(valid_x, valid_y, color='orange', label='Полином Лагранжа (кусочно)')
        plt.title(' ... ')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.ylim(-1, 1)
        plt.grid()
        plt.legend()
        plt.show()
```

Листинг 2: Визуализация интерполяционного полинома Лагранжа

Данный фрагмент кода визуализирует результат работы интерполяции, где синие точки – исходные данные, а оранжевая кривая – результат интерполяции полиномом Лагранжа, построенный кусочно. Такой подход позволяет оценить поведение полинома на отдельных интервалах и выявить аномалии, если они возникают.

4.3 Код интерполяции с использованием кубического сплайна

Этот раздел посвящен реализации интерполяции кубическим сплайном с заданием краевых условий. Кубические сплайны обеспечивают плавную интерполяцию данных за счет использования кусочно-заданных кубических полиномов, которые непрерывны по значению функции, её первой и второй производным. Это делает их особенно полезными для задач, требующих высокой гладкости кривой.

```
import numpy as np

left_derivative = (unique_y[1] - unique_y[0]) / (unique_x[1] - unique_x[0])
right_derivative = (unique_y[-1] - unique_y[-2]) / (unique_x[-1] - unique_x[-2])

def cubic_spline(x, y, left_derivative, right_derivative):
n = len(x) - 1
h = np.diff(x)
alpha = np.zeros(n)

for i in range(1, n):
alpha[i] = (3 / h[i]) * (y[i + 1] - y[i]) - (3 / h[i - 1]) * (y[i] - y[i - 1])

alpha[0] = 3 * ((y[1] - y[0]) / h[0] - left_derivative)
alpha[n-1] = 3 * (right_derivative - (y[n] - y[n-1]) / h[n-1])

l = np.zeros(n + 1)
mu = np.zeros(n)
z = np.zeros(n + 1)

l[0] = 2 * h[0]
mu[0] = 0.5
z[0] = alpha[0] / l[0]

for i in range(1, n):
l[i] = 2 * (x[i + 1] - x[i - 1]) - h[i - 1] * mu[i - 1]
mu[i] = h[i] / l[i]
z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i]

l[n] = h[n - 1] * (2 - mu[n - 1])
z[n] = (alpha[n - 1] - h[n - 1] * z[n - 1]) / l[n]

c = np.zeros(n + 1)
b = np.zeros(n)
d = np.zeros(n)

c[n] = z[n]

for j in range(n - 1, -1, -1):
c[j] = z[j] - mu[j] * c[j + 1]
b[j] = (y[j + 1] - y[j]) / h[j] - h[j] * (c[j + 1] + 2 * c[j]) / 3
d[j] = (c[j + 1] - c[j]) / (3 * h[j])
```

```
return b, c, d

b, c, d = cubic_spline(unique_x, unique_y, left_derivative, right_derivative)

def spline_interpolation(x, y, b, c, d, x_val):
n = len(x) - 1
for i in range(n):
if x[i] <= x_val <= x[i + 1]:
dx = x_val - x[i]
return y[i] + b[i] * dx + c[i] * dx**2 + d[i] * dx**3
return None

x_interpolated = np.linspace(min(unique_x), max(unique_x), 130)
y_spline_interpolated = [spline_interpolation(unique_x, unique_y, b, c, d, xi) for xi in x_interpolated]

plt.figure(figsize=(12, 8))
plt.scatter(unique_x, unique_y, color='blue', label='Исходнүе точки', s=10)
plt.plot(x_interpolated, y_spline_interpolated, color='green', label='Кубическиі сплаін')
plt.title(' ... ')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.legend()
plt.show()
```

Листинг 3: Интерполяция кубическим сплайном с краевыми условиями

Теоретическое обоснование кубического сплайна. Кубический сплайн строится из полиномов третьей степени $S_i(x)$ на каждом интервале $[x_i, x_{i+1}]$, таких что:

$$S_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

Гладкость сплайна достигается за счет требований непрерывности функции $S(x)$, её первой производной $S'(x)$ и второй производной $S''(x)$ на концах каждого интервала. Это создает систему уравнений для коэффициентов b_i, c_i, и d_i, которая решается методом прогонки.

Настройка краевых условий. Краевые условия включают первую производную на концах интервала. Для этого мы приближенно вычисляем производные на краях данных:

$$\texttt{left\_derivative} = \frac{y[1] - y[0]}{x[1] - x[0]}, \quad \texttt{right\_derivative} = \frac{y[-1] - y[-2]}{x[-1] - x[-2]}. \tag{4}$$

Эти значения устанавливают наклон кривой в начале и конце, что позволяет контролировать поведение сплайна на границах.

Разбор кода функции `cubic_spline`. Функция `cubic_spline` вычисляет коэффициенты b, c, и d для каждого отрезка между узловыми точками. Она выполняется следующим образом:

- **Вычисление разностей** h. Разности $h[i] = x[i+1] - x[i]$ представляют длины интервалов между соседними точками и используются для определения наклонов на каждом интервале.
- **Формирование правой части** α. Параметры $\alpha[i]$ вычисляются для построения системы уравнений, учитывающей изменения в значениях y и длинах интервалов h. Внутренние значения $\alpha[i]$ задаются как:

  $$\alpha[i] = \frac{3}{h[i]}(y[i+1] - y[i]) - \frac{3}{h[i-1]}(y[i] - y[i-1]).$$

  Для учета производных на краях применяются модифицированные формулы для $\alpha[0]$ и $\alpha[n-1]$:

  $$\alpha[0] = 3\left(\frac{y[1] - y[0]}{h[0]} - \texttt{left\_derivative}\right),$$
  $$\alpha[n-1] = 3\left(\texttt{right\_derivative} - \frac{y[n] - y[n-1]}{h[n-1]}\right).$$

- **Решение системы уравнений**. Система уравнений решается с использованием трех диагональных матриц. Коэффициенты l, μ, и z на каждом шаге i задаются следующим образом:

  $$l[i] = 2(x[i+1] - x[i-1]) - h[i-1]\mu[i-1], \quad \mu[i] = \frac{h[i]}{l[i]}, \quad z[i] = \frac{\alpha[i] - h[i-1]z[i-1]}{l[i]}.$$

  Эти параметры позволяют найти значения c.

- **Вычисление коэффициентов** b, c, и d. Получив c, вычисляем остальные коэффициенты:

  $$b[i] = \frac{y[i+1] - y[i]}{h[i]} - \frac{h[i](c[i+1] + 2c[i])}{3}, \quad d[i] = \frac{c[i+1] - c[i]}{3h[i]}.$$

  Эти коэффициенты позволяют задать кубический полином на каждом интервале, который гарантирует гладкость функции.

Функция `spline_interpolation`. Эта функция используется для вычисления значения интерполированной функции в любой точке x_{val} с использованием коэффициентов сплайна. Она находит интервал, которому принадлежит x_{val}, и применяет кубический полином для вычисления значения:

$$S(x_{\text{val}}) = y[i] + b[i](x_{\text{val}} - x[i]) + c[i](x_{\text{val}} - x[i])^2 + d[i](x_{\text{val}} - x[i])^3.$$

Преимущества и особенности кубического сплайна. Кубический сплайн позволяет получить гладкую интерполяцию, которая подходит для моделирования данных с высокой точностью. Использование производных на краях интервала позволяет контролировать форму кривой на границах, избегая чрезмерного изгиба. Это делает кубические сплайны полезными в задачах, требующих гладкости второй производной, например, при моделировании физических процессов.

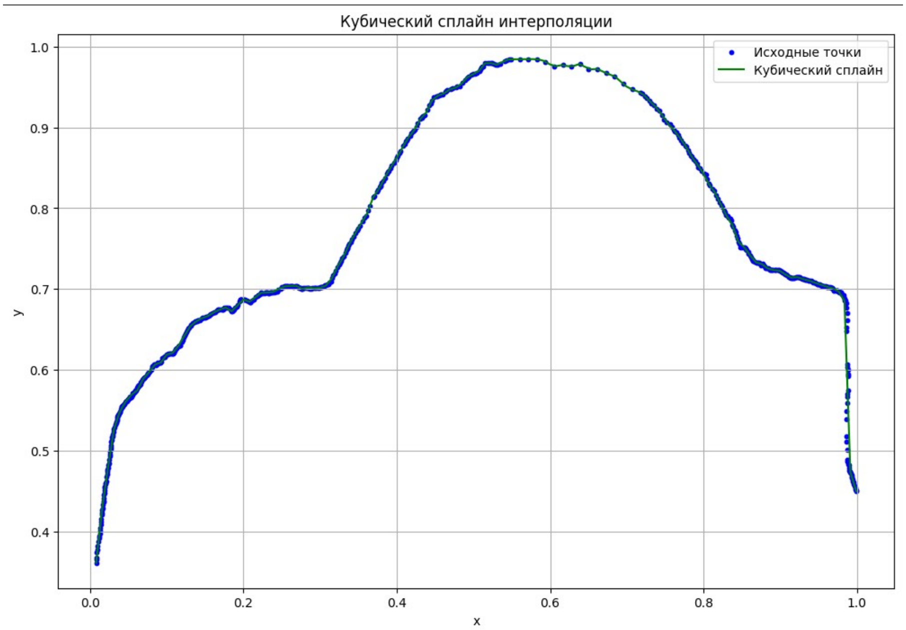

Рис. 3: Кубический сплайн для данных контура автомобиля

Рисунок иллюстрирует результат построения кубического сплайна, который является более гибким и позволяет избежать резких колебаний между точками, обеспечивая гладкость на каждом интервале между узловыми точками.

4.4 Анализ производных кубического сплайна

Для углубленного анализа формы кривой были также построены графики первых и вторых производных сплайна. Производные сплайна помогают лучше понять поведение кривой, показывая, как изменяется наклон и кривизна на каждом участке.

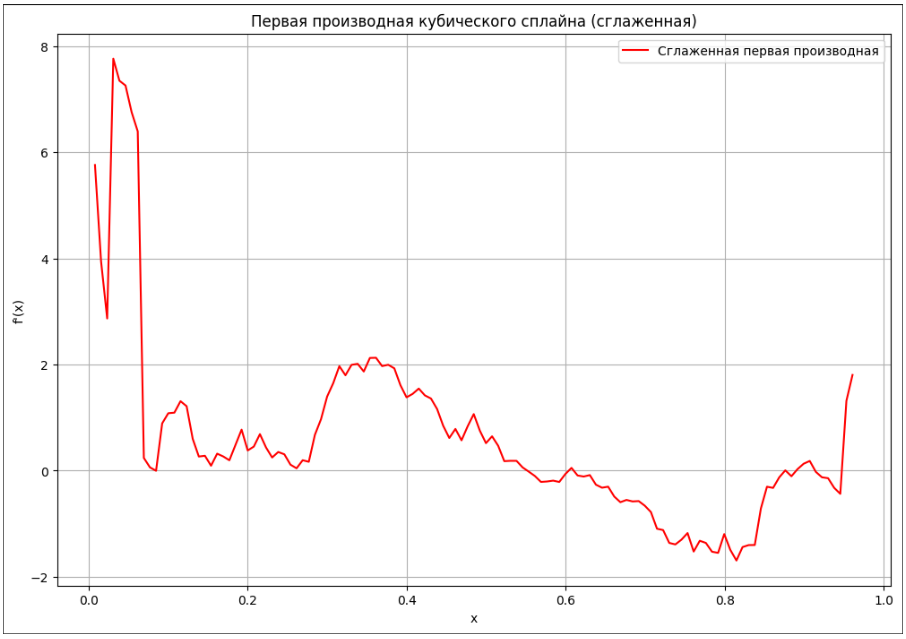

Рис. 4: Первая производная кубического сплайна

На Рисунке 3 показана первая производная кубического сплайна, которая демонстрирует изменения наклона на каждом участке интерполяции. Плавные изменения первой производной говорят о гладкости переходов между узловыми точками.

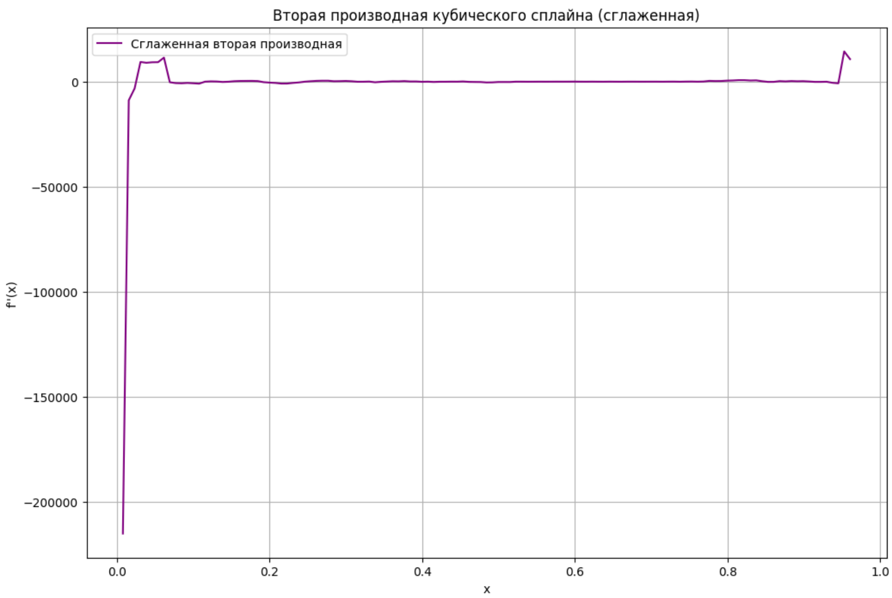

Рис. 5: Вторая производная кубического сплайна

Рисунок 4 иллюстрирует вторую производную кубического сплайна, которая показывает кривизну графика. Плавность второй производной указывает на отсутствие резких изменений в кривизне, что обеспечивает визуально приятную и гладкую интерполяцию контура автомобиля.

5 Заключение

В данной лабораторной работе были реализованы и исследованы методы интерполяции Лагранжа и кубического сплайна. Метод Лагранжа позволяет построить полином, проходящий через все заданные точки, однако может быть чувствительным к количеству точек и их расположению. Кубический сплайн, в свою очередь, обеспечил более гибкую интерполяцию с плавными переходами между точками и контролем кривизны благодаря краевым условиям.

Эти методы интерполяции находят применение в задачах, связанных с моделированием кривых, например, при обработке контуров объектов, как в нашем случае. Кубический сплайн показал свою эффективность для создания гладкого контура автомобиля, что может быть полезно для задач графического моделирования и компьютерного зрения.