Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени
Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехники и комплексной автоматизации»
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине «Вычислительная математика»

Студент:	Гавриш Александр Александрович
Группа:	РК6-51Б
Тип задания:	лабораторная работа
Тема:	Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование

Студент ______________ Гавриш А.А.
подпись, дата Фамилия, И.О.

Преподаватель ______________ ______________
подпись, дата Фамилия, И.О.

Москва, 2023

Содержание

Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование **3**
1 Задание . . . 3
2 Цель выполнения лабораторной работы . . . 5
3 Выполненные задачи . . . 5
4 Выбор произвольной области множества Мандельброта и построение контура . . . 6
5 Разработка функций для загрузки и визуализации множества точек из файла . . . 7
6 Создание разреженного множества интерполяционных узлов . . . 8
7 Разработка функции вычисления коэффициентов кубического сплайна . . . 9
8 Разработка функции для вычисления расстояния . . . 12
9 Построение аппроксимации по заданным узлам с помощью кубического сплайна . . . 13
10 Разработка функции для выполнения базовой части . . . 14
11 Разработка класса для автоматического вычисления производной функции 15
12 Разработка функции автоматического расчета первой производной кубического сплайна . . . 15
13 Разработка функции построения нормали к заданному вектору . . . 16
14 Построение векторов производных и нормалей . . . 17
15 Заключение . . . 17
Заключение . . . 17

Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование

1 Задание

Множество Мандельброта (фрактал) – удивительное явление широко извест- ное за пределами соответствующей области математики благодаря бесконечному многообразию форм и ярким визуализациям.

Определение 1

Точки фрактала $c \subset \mathbb{C}$ в пространстве комплексных чисел определяется реккуретной формулой, задающей последовательность, ограниченную для точек принадлежащих фракталу:

$$\begin{aligned} &\forall c \in \mathbb{C}, \exists Z \in \mathbb{R} : |z_i| < Z,\ i \in \mathbb{N}, \\ &z_i = z_{i-1}^2 + c, \\ &z_0 = 0 \end{aligned} \tag{1}$$

Другими словами, для точек фрактала, независимо от длины последовательности при $i \to \infty$, $|z_i|$ не превысит некоторого заранее заданного действительного числа. Для вычислительнего эксперимента следует положить $Z = 1000$, $i \in [0 : 255]$.

В данной работе потребуется интерполировать некоторый фрагмент границы фрактала c, описываемой в параметрическом виде неизвестными функциями $x(t)$, $yt(t)$, на основе его отдельных точек, используя параметрически задаваемый кубический сплайн.

Базовая часть:

1. Используя заранее подготовленный скрипт, выбрать произвольную область множества Мандельброта и построить фрагмент его границы (контура), сформи- ровав файл contours.txt. Использование неуникального файла contours.txt считается списыванием, равно как и использование чужого кода.

Файл contours.txt содержит упорядоченную последовательность точек на плоскости $P = \{(x_i, y_i)\}_{i=1}^{N}$, принадлежащих выбранному фрагменту границы фрактала c. Сопоставляя каждой паре координат естественную координату t, предполагать, что $x_i = x(t_i), y_i = y(t_i)$. Выбранный контур должен содержать по меньшей мере 100 точек (100 строк в файле contours.txt).

2. Разработать код для загрузки и визуализации множества точек P из файла contours.txt.

3. Задать разреженное множество интерполяционных узлов $\hat{P} = \{(x_j, y_j)\}_{i=1}^{N}$, $\hat{N} = \lfloor N/M \rfloor$, $j = M \times i, \hat{P} \subset P$. Положить $M = 10$.

4. По каждому измерению найти коэффициенты естественного параметрического кубического сплайна a_{jk} и b_{jk}, путём решения соответствующих разрешающих СЛАУ,

в результате должен получится сплайн вида:

$$
\begin{aligned}
&\tilde{x}(t) = \textstyle\sum_{j=1}^{\hat{N}-1} I_j(t)(a_{j0} + a_{j1}(t-t_j) + a_{j2}(t-t_j)^2 + a_{j3}(t-t_j)^3),\\
&\tilde{y}(t) = \textstyle\sum_{j=1}^{\hat{N}-1} I_j(t)(b_{j0} + b_{j1}(t-t_j) + b_{j2}(t-t_j)^2 + b_{j3}(t-t_j)^3),\\
&I_j(t) = \begin{cases} 1, & \text{если } t \in [t_j, t_{j+1}) \\ 0, & \text{в противном случае} \end{cases}
\end{aligned}
\tag{2}
$$

где $I_j(t)$ - индикаторная функция принадлежности интервалу.

5. Вычислить расстояния $\rho[(\tilde{x}(t_i), \tilde{y}(t_i)), (x(t_i), y(t_i))]$ и представить вывод (среднее и стандартное отклонение) в отчёте.

6. Отобразить в отчёте полученный сплайн используя $t \in [0, t_N]$ с частым шагом $h = 0.1$ совместно с исходным множеством точек P, а также узловыми точками $\hat{P}$. С чем связана наблюдаемая ошибка интерполяции? Как её можно уменьшить? Вывод следует привести в отчёте.

7. В результате выполнения базовой части задания, помимо прочих, должна быть разработана функция lab1_base(filename_in:str, factor:int, filename_out:str), где filename_in - входной файл contours.txt, factor - значение параметра M, filename_out - имя файла результата (как правило coeffs.txt), содержащего коэффициенты a_{jk} и b_{jk} в виде матрицы $\hat{N}-1$ строк на 8 столбцов. Функция lab1_base должна реализовывать базовую часть задания.

Продвинутая часть:

8. Используя концепцию дуальных чисел $v = a + \epsilon b, \epsilon^2 = 0$, и перегрузку операторов сложения и умножения в Python, необходимо реализовать класс *AutoDiffNum*, для автоматического вычисления производной некоторой функции.

9. Реализовать функцию автоматического расчёта первой производной кубического сплайна $G(t) = \frac{d}{dt}(\tilde{x}(t), \tilde{y}(t))$.

10. Реализовать функцию построения нормали $R(t_j)$ к заданному вектору $G(t_j)$.

11. Построить векторы $G(t_j)$ и $R(t_j)$ в соответствующих точках сплайна, выбрав наглядную частоту прореживания (не менее 5 точек на контур) и масштаб.

12. Опциональное задание. Для каждой пропущенной точки (x_i, y_i) исходного контура найти (численно) ближайшую на соответствующем участке сплайна. Фактически нужно решить оптимизационную задачу:

$$t^* = \underset{t \in [0, t_N]}{argmin} \sqrt{(\tilde{x}(t) - x_i)^2 + (\tilde{y}(t) - y_i)^2} \tag{3}$$

К примеру, используя простейший метод деления отрезка пополам (дихотомии) до 10 итераций. В отчёте необходимо отобразить на графике соответствующие точки, выбрав наглядную частоту прореживания и привести среднюю оценку погрешности. Сравнить результаты с вычисленными ранее значениями $\rho[(\tilde{x}(t_i), \tilde{y}(t_i)), (x(t_i), y(t_i))]$. Все полученные в работе изображения, включаемые в отчёт, должны быть сохранены в векторном формате (PDF или EPS) и размещены рядом с исходным кодом разработанной программы.

2 Цель выполнения лабораторной работы

Цель выполнения лабораторной работы: освоение методов интерполяции на базе параметрически заданных кубических сплайнов и практическое применение автоматического дифференцирования с использованием концепции дуальных чисел.

3 Выполненные задачи

1. Используя скрипт для генерации множества точек фрагмента контура множества Мандельброта, был сформирован файл contours.txt.

2. На языке программирования Python были написаны две функция: первая загружает множество точек из файла contours.txt, вторая визуализирует точки множества на плоскости.

3. На языке программирования Python была написана функция для задания разреженного множества интерполяционных узлов с заданным шагом.

4. На языке программирования Python была написана функция, которая вычисляет коэффиценты естественного кубического сплайна, путём решения разрешающих СЛАУ.

5. На языке программирования Python была написана функция для вычисления расстояний между наборами множества точек и нахождения среднего и стандартного отклонения.

6. Была проведена и продемонстрирована на графике аппроксимация контура множества Мандельброта с помощью кубических сплайнов по точкам разреженного множества с заданным частым шагом.

7. На языке программирования Python была реализована функция `lab1_base`, которая, помимо выполнения предыдущих пунктов, сохраняет коэффициенты кубического сплайна в файл.

8. На языке программирования Python реализован класс *AutoDiffNum* для автоматического вычисления производной некоторой функции, используя концепцию дуальных чисел и перегрузку операторов умножения, сложения и возведения в степень.

9. На языке программирования Python была написана функция, которая автоматически рассчитывает первую производную кубического сплайна.

10. На языке программирования Python реализована функция построения нормали к заданному вектору.

11. Были построены векторы производной и нормали в точках разреженного множества с наглядной частотой прореживания и масштабом.

4 Выбор произвольной области множества Мандельброта и построение контура

Для выбора фрагмента множества Мандельброта запустим заранее подготовленный скрипт для генерации и визуализации точек одноименного множества. Используя инструмент приближения (лупу), выберем произвольный участок множества (рис. 1) и завершим работу скрипта.

После завершения работы программы в качестве результата имеем файл `contours` формата `txt`, который содержит упорядоченный список координат точек на плоскости $P = \{(x_i, y_i)\}_{i=1}^{N}$, принадлежащих выбранному фрагменту границы фрактала. Сопоставляя каждой паре координат естественную координату $t \in [0, 2524]$, предполагаем, что $x_i = x(t_i)$, $y_i = y(t_i)$. То есть, t_i - множество индексов контура множества Мандельброта.

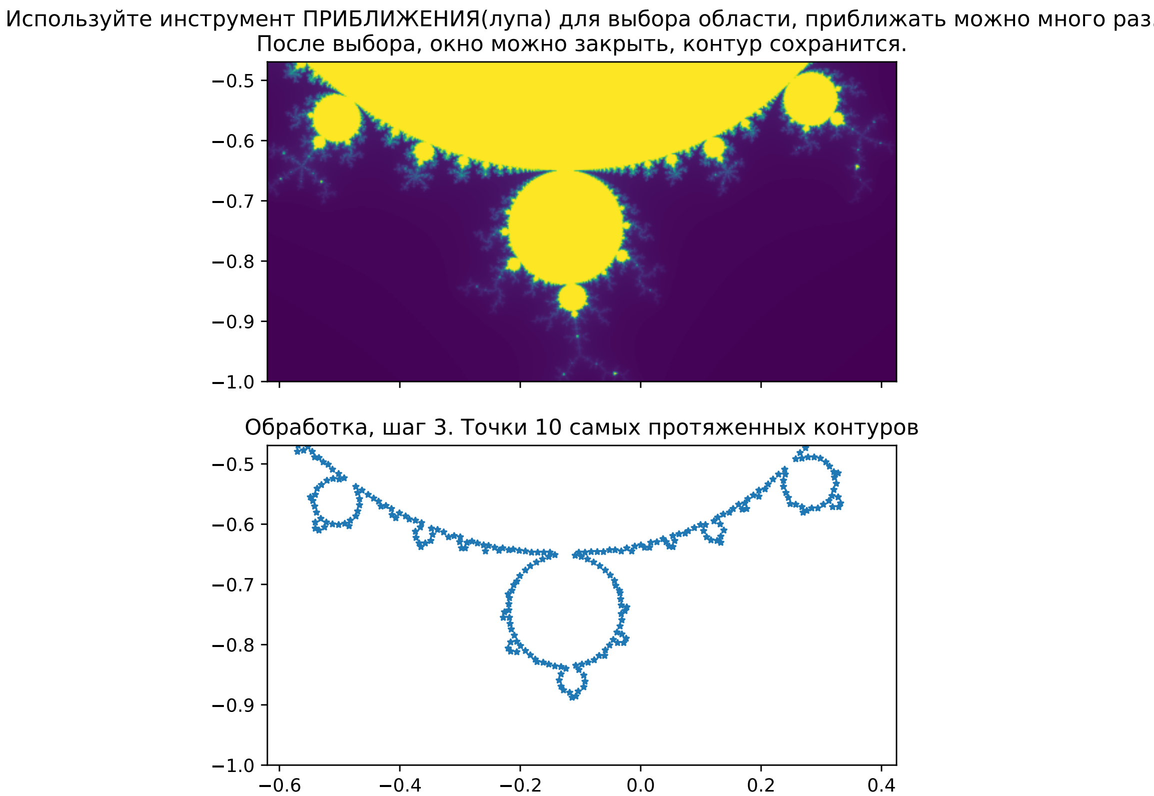

Рис. 1. Контур множества Мандельброта

5 Разработка функций для загрузки и визуализации множества точек из файла

Для загрузки и визуализации множества точек P из файла реализуем две функции (листинг 1): первая будет загружать данные из файла, вторая визуализировать полученные точки на плоскости.

Листинг 1. Загрузка и визуализация точек

```
def read_points(file_name: str) -> Tuple[np.int32, np.ndarray, np.ndarray]:
    points = np.loadtxt(file_name, dtype=np.float64)
    x, y = points[:, 0], points[:, 1]
    return len(points), x, y

def display_points(x: np.ndarray, y: np.ndarray,
                   ax: plt.Axes, settings: dict) -> None:
    ax.scatter(x, y, **settings)
```

Функция `read_points` принимает на вход название файла из которого необходимо считать множество точек. Загрузка точек происходит с использованием функции np.loadtxt, которая возвращает двумерный массив координат точек с плавающей запятой. Этот массив разделяется на два одномерных: x и y, которые содержат x и y координаты всех точек соответственно. Для дальнейшей работы с полученными значениями функция возвращает длину множества и два массива с координатами точек.

Функция display_points визуализирует точки на заданной координатной плоскости с использованием метода scatter из библиотеки matplotlib. Разработанная функция принимает массивы, которые характеризуют x и y координаты точек для отображения, а также словарь settings для настройки внешнего вида точек (например цвет. размер).

Визуализация точек множества P приведена на (рис. 2)

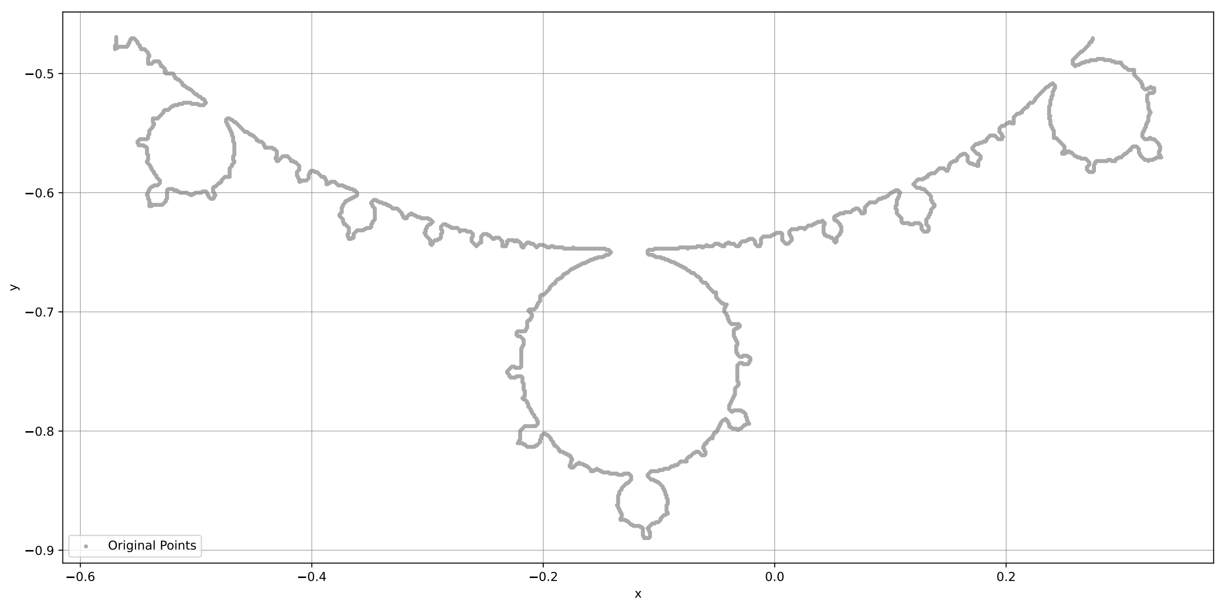

Рис. 2. Множество точек контура множества Мандельброта

6 Создание разреженного множества интерполяционных узлов

Для создания разреженного множества интерполяционных узлов $\hat{P} = \{(x_j, y_j)\}_{j=1}^{\hat{N}}$, где $\hat{N} = \lfloor N/M \rfloor$, дополнительно сформируем массив индексов в диапазоне $[0, N]$ с заданным шагом $M = 10$.

Затем применяем слайсинг с этим массивом индексов к исходному массиву координат, который представляет собой выбранный контур множества Мандельброта. В результате формируется разреженное множество $\hat{P}$ (рис. 3). Реализация данного метода представлена на листинге 2.

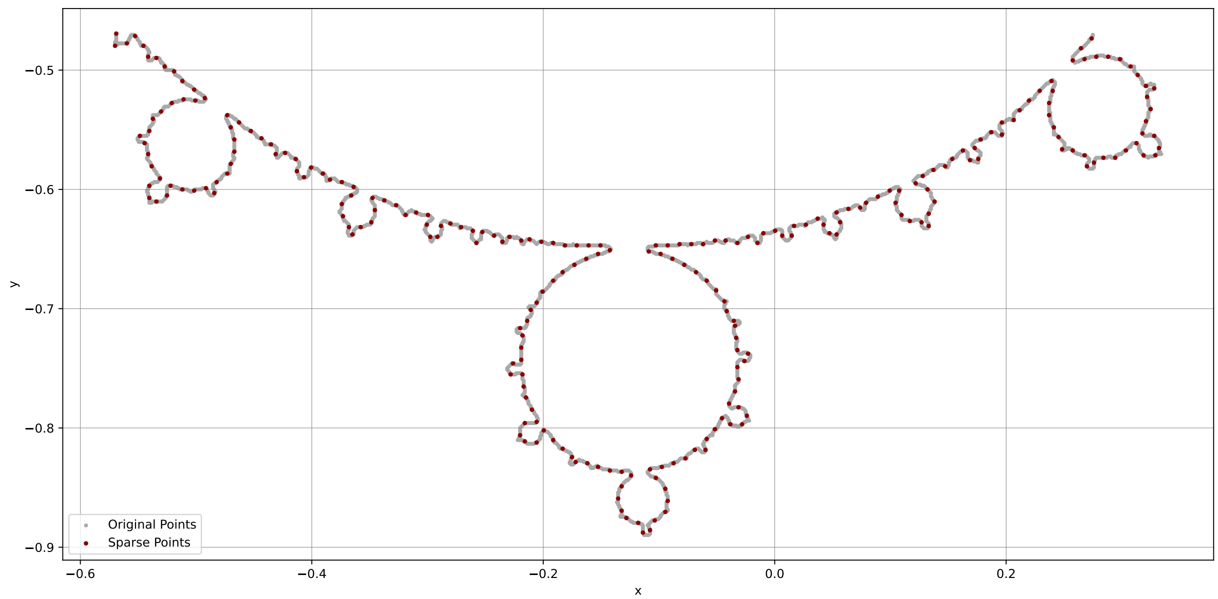

Рис. 3. Разреженное множество

Листинг 2. Создание разреженного множества

```
def create_sparse_set(length: np.int32, M: np.int32,
                      x: np.ndarray, y: np.ndarray) -> None:
    sparse_set = np.arange(0, length, M)
    x_sparse, y_sparse = x[sparse_set], y[sparse_set]
    return sparse_set, x_sparse, y_sparse

sparse_set, x_sparse, y_sparse = create_sparse_set(length, 10, x, y)
```

7 Разработка функции вычисления коэффициентов кубического сплайна

Кубический сплайн является примером локальной интерполяции, что будет видно далее из его определения, которое понадобится нам для вычисления его коэффициентов.

Определение 2
Пусть функция $f(x)$ задана в n интерполяционных узлах $a = x_1, x_2, \dots, x_n = b$ на отрезке $[a; b]$. Тогда кубическим сплайном для функции $f(x)$ называется функция $S(x)$, для которой верно:

1. $S(x)$ кусочно задана кубическими многочленами $S_i(x)$ на каждом отрезке $[x_i; x_{i+1}]$, $i = 1, \dots, n-1$;

2. $S_i(x_i) = f(x_i)$ и $S_i(x_{i+1}) = f(x_{i+1})$, $i = 1, \dots, n-1$;

3. сопряжение значений смежных многочленов: $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$, $i = 1, \dots, n-2$;

4. сопряжение первых производных: $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$, $i = 1, \ldots, n-2$;

5. сопряжение вторых производных: $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$, $i = 1, \ldots, n-2$;

6. заданы граничные условия:

   (a) естественные граничные условия: $S''(x_1) = S''(x_n) = 0$;

   (b) граничные условия на касательную: $S'(x_1) = f'(x_1)$ и $S'(x_n) = f'(x_n)$;

Кубический полином задается 4 константами, поэтому для кубического сплайна нам необходимо вычислить $4(n-1)$ коэффициентов:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{4}$$

На данном этапе можно заметить, что подход локальной интерполяции, в частности кубическим сплайном, имеет важное преимущество перед глобальной: увеличение количества узлов не ведет к увеличению степени полинома – он всегда остается полиномом третьей степени.

Из формулы (4) видно, что $S_i(x_i) = a_i = f(x_i)$. Далее, как было показано в лекциях [1], можно получить следующие рекуррентные формулы для b_i и d_i, исходя из определения кубического сплайна.

$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(c_{i+1} + 2c_i), \tag{5}$$

$$d_i = \frac{c_{i+1} - c_i}{3h_i}, \tag{6}$$

где $h_i = x_{i+1} - x_i$.

В итоге для нахождения векторов коэффициентов b_i и d_i необходимо найти вектор коэффициентов c_i. Используя формулы из лекций, запишем рекуррентное уравнение относительно c_{i-1}, c_i и c_{i+1}:

$$h_{i-1}c_{i-1} + 2(h_i + h_{i-1})c_i + h_i c_{i+1} = \frac{3}{h_{i-1}}(a_i - a_{i-1}) - \frac{3}{h_{i-2}}(a_{i-1} - a_{i-2}). \tag{7}$$

Тогда коэффициенты кубического сплайна можно получить, решая следующее матричное уравнение $\mathbf{Ac} = \mathbf{b}$, $\mathbf{c} = [c_1, ..., c_n]^T$:

$$
\begin{pmatrix}
1 & 0 & \dots & \dots & \dots & 0 \\
h_1 & 2(h_2+h_1) & h_2 & 0 & \dots & 0 \\
0 & h_2 & 2(h_3+h_2) & h_3 & \dots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \dots & 0 & h_{n-2} & 2(h_{n-2}-h_{n-1}) & h_{n-1} \\
0 & \dots & \dots & \dots & \dots & 1
\end{pmatrix}
\begin{pmatrix}
c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \\ c_n
\end{pmatrix} =
= \begin{pmatrix}
0 \\
\frac{3}{h_2}(a_3-a_2) - \frac{3}{h_1}(a_2-a_1) \\
\frac{3}{h_3}(a_4-a_3) - \frac{3}{h_2}(a_3-a_2) \\
\vdots \\
\frac{3}{h_{n-1}}(a_n-a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1}-a_{n-2}) \\
0
\end{pmatrix}, \tag{8}
$$

где $h_i = t_{i+1} - t_i$.

Численное решение матричного уравнения (8) находится используя метод `numpy.linalg.inv` из библиотеки `Numpy`. Функция, вычисляющая коэффициенты кубического сплайна, приведена на листинге 3.

Листинг 3. Коэффициенты сплайна

```
def find_coefficients(sparse_set: np.ndarray, sparse_values: np.ndarray) \
        -> Tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    h = np.diff(sparse_set)

    A_matrix = np.zeros((len(sparse_set), len(sparse_set)))
    A_matrix[0][0], A_matrix[-1][-1] = 1, 1

    for index in range(1, len(sparse_set) - 1):
        A_matrix[index][index-1] = h[index-1]
        A_matrix[index][index] = 2 * (h[index] + h[index-1])
        A_matrix[index][index+1] = h[index]

    B_matrix = np.zeros(len(sparse_set))

    for index in range(1, len(sparse_set) - 1):
        B_matrix[index] = 3/h[index] * \
            (sparse_values[index+1] - sparse_values[index]) - \
            3/h[index-1] * (sparse_values[index] - sparse_values[index-1])

    c_coef = np.linalg.solve(A_matrix, B_matrix)
    d_coef = np.array([(c_coef[index+1] - c_coef[index])/(3*h[index])
                       for index in range(len(sparse_set) - 1)])
    a_coef = np.array([sparse_values[index]
                       for index in range(len(sparse_set))])
    b_coef = np.array([(1/h[index]) *
                        (a_coef[index+1] - a_coef[index]) - (h[index]/3) *
                        (c_coef[index+1] + 2*c_coef[index])
                        for index in range(len(sparse_set) - 1)])
    return a_coef[:-1], b_coef, c_coef[:-1], d_coef
```

8 Разработка функции для вычисления расстояния

Для вычисления расстояний между интерполированными точками $\tilde{x}(t_i), \tilde{y}(t_i)$ и исходными точками $x(t_i), y(t_i)$ применим евклидову метрику, которая задаётся формулой $\rho = \sqrt{(x - \tilde{x})^2 + (y - \tilde{y})^2}$.

Листинг 4. Нахождение расстояний

```
def calculate_distances(val: np.ndarray, val_inter: np.ndarray,
                        length: np.int32) -> None:
    distances = np.sqrt((val[0] - val_inter[0][:length]) **
                        2 + (val[1] - val_inter[1][:length]) ** 2)
    axs[0][1].plot(distances, **settings_for_plot['Distances'], label=(
        f'Среднее отклонение: {np.mean(distances):.10f}\n'
        f'Стандартное отклонение: {np.std(distances):.10f}'))
    print(f'Mean deviation: {np.mean(distances):.10f}')
    print(f'Standard deviation: {np.std(distances):.10f}')
```

Функция `calculate_distances` (листинг 4) принимает на вход два массива: val и val_inter. Массив val содержит координаты исходных точек, а val_inter - координаты интерполированных точек. Также функция принимает параметр length, который определяет, сколько точек следует анализировать. Внутри функции вычисляются расстояния между соответствующими точками и сохраняются в массиве distances.

После вычисления расстояний производим анализ полученных данных: вычисляем среднее и стандартное отклонения (рис. 4).

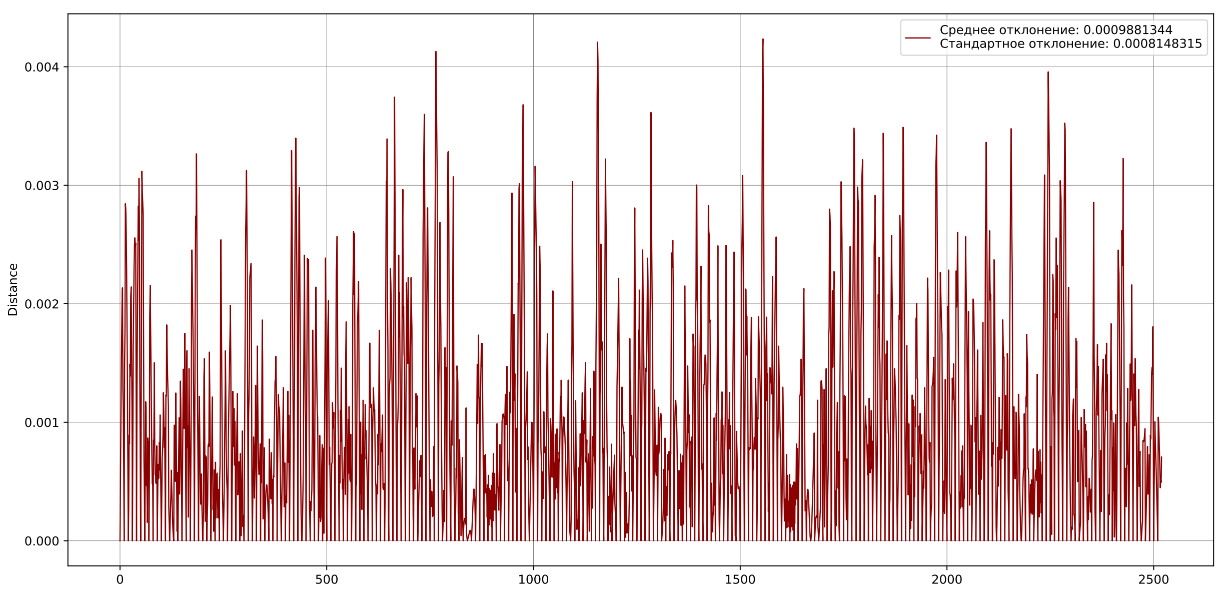

Рис. 4. График расстояний

Среднее отклонение $= 0.0009881344$
Стандартное отклонение $= 0.0008148315$

9 Построение аппроксимации по заданным узлам с помощью кубического сплайна

На данном этапе мы уже имеем все необходимые функции для построения графика сплайна по заданным узлам. Вычислим вектора коэффициентов и сгенерируем множество точек $t \in [0, t_N]$ с шагом $h = 0.1$, и по этому множеству вычислим значения сплайна. Воспользуемся модулем `matplotlib.pyplot` и построим аппроксимацию по точкам разреженного множества в виде графика (рис. 5).

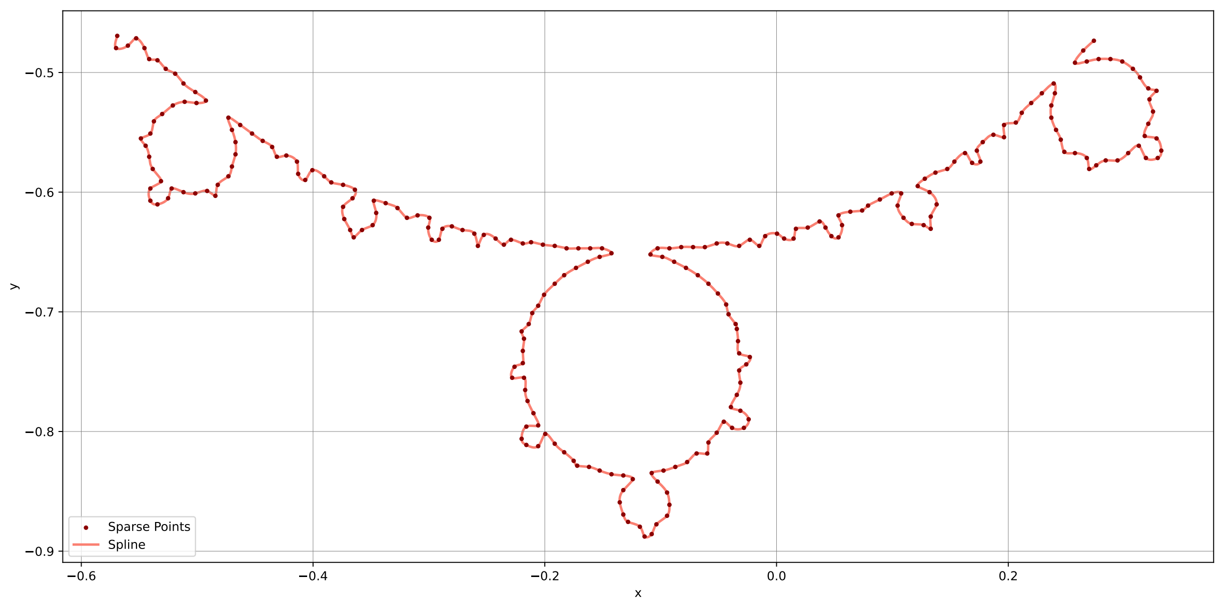

Рис. 5. Кубический сплайн и интерполяционные узлы

Наблюдаемая ошибка интерполяции может быть связана с малым количеством узлов $\hat{P}$, а также со сложной формой контура множества Мандельброта.

Для уменьшения ошибки можно воспользоваться следующими решениями:

– увеличить число узловых точек $\hat{P}$: это может помочь в получении более точного сплайна, особенно на участках с высокой кривизной.

– использовать более точные методы интерполяции,

– предварительно обработать данные: сглаживание или фильтрация исходных данных для уменьшения шума.

– использовать адаптивный выбор узлов: вместо равномерного распределения узлов можно использовать адаптивный метод, который учитывает кривизну функции.

10 Разработка функции для выполнения базовой части

Функция `lab1_base` (листинг 5) служит для реализации базовой части задания (все разработанные ранее функции инкапсулированы внутри реализуемой функции) и сохранения коэффициентов кубического сплайна и принимает три аргумента:

1. `filename_in` — имя входного файла, обычно `contours.txt`, содержащего исходные точки.

2. `factor` — значение параметра M, который используется для создания разреженного множества узлов.

3. `filename_out` — имя выходного файла, обычно `coeffs.txt`, в который сохраняются коэффициенты a_{jk} и b_{jk}.

Листинг 5. Функция для выполнения базовой части

```
def lab1_base(filename_in: str, factor: int, filename_out: str) \
        -> Tuple[np.ndarray, np.ndarray[4], np.ndarray[4]]:
    length, x, y = read_points(filename_in)
    sparse_set, x_sparse, y_sparse = create_sparse_set(length, factor, x, y)

    x_coeffs = find_coefficients(sparse_set, x_sparse)
    y_coeffs = find_coefficients(sparse_set, y_sparse)

    h = 0.1
    x_inter, y_inter = calculate_spline(sparse_set, x_coeffs, y_coeffs, h)

    display_points(x, y, axs[0][0], settings={
                   **settings_for_plot['OriginalPoints']})

    current_axes = [axs[0][0], axs[1][0], axs[1][1]]
    for ax in current_axes:
        display_points(x_sparse, y_sparse, ax, settings={
            **settings_for_plot['SparsePoints']})

    calculate_distances((x_inter[::10], y_inter[::10]),
                        (x, y), length//factor*factor)

    axs[1][0].plot(x_inter, y_inter, **settings_for_plot['Spline'])
    axs[1][1].plot(x_inter, y_inter, **settings_for_plot['Spline'])

    np.savetxt(filename_out, np.c_[x_coeffs[0], x_coeffs[1],
                                   x_coeffs[2], x_coeffs[3],
                                   y_coeffs[0], y_coeffs[1],
                                   y_coeffs[2], y_coeffs[3]],
               delimiter=' ', fmt='%+.10e')

    return sparse_set, x_coeffs, y_coeffs
```

Результатом работы функции являются соответствующие коэффициенты a_{jk} и b_{jk}, которые сохраняются в выходной файл.

11 Разработка класса для автоматического вычисления производной функции

Для реализации класса `AutoDiffNum` (листинг 6), который использует концепцию дуальных чисел ($v = a + \varepsilon b$, $\varepsilon^2 = 0$) для автоматического вычисления производной функции, необходимо перегрузить следующие операторы:

– оператор сложения `__add__`: $\langle a, b\rangle + \langle c, d\rangle = \langle a + c, b + d\rangle$

– оператор вычитания `__sub__`: $\langle a, b\rangle - \langle c, d\rangle = \langle a - c, b - d\rangle$

– оператор умножения `__mul__`: $\langle a, b\rangle \times \langle c, d\rangle = \langle a \cdot c, b \cdot c + a \cdot d\rangle$

– оператор возведения в степень `__pow__`: $\langle a, b\rangle^n = \langle a^n, n \cdot a^{n-1} \cdot b\rangle$

Листинг 6. Автоматическое вычисление производной

```
class AutoDiffNum:
    def __init__(self, a, b):
        self.real = a
        self.dual = b

    def __repr__(self):
        if self.dual < 0:
            return f'{self.real} - {-self.dual}e'
        return f'{self.real} + {self.dual}e'

    def __add__(self, other):
        if isinstance(other, (int, float)):
            return AutoDiffNum(self.real + other, self.dual)
        return AutoDiffNum(self.real + other.real, self.dual + other.dual)

    def __sub__(self, other):
        if isinstance(other, (int, float)):
            return AutoDiffNum(self.real - other, self.dual)
        return AutoDiffNum(self.real - other.real, self.dual - other.dual)

    def __mul__(self, other):
        if isinstance(other, (int, float)):
            return AutoDiffNum(self.real * other, self.dual * other)
        return AutoDiffNum(self.real * other.real, self.dual * other.real + self.real * other.dual)

    def __pow__(self, power):
        return AutoDiffNum(self.real ** power, power * self.real ** (power - 1) * self.dual)
```


12 Разработка функции автоматического расчета первой производной кубического сплайна

Реализация функции для автоматического расчета первой производной кубического сплайна $G(t) = \frac{d}{dt}(\tilde{x}(t), \tilde{y}(t))$ (листинг 7) основана на работе ранее разработанного класса *AutoDiffNum*, вычисляя значения кубического сплайна в заданном множестве точек, координаты которых представлены в виде дуальных чисел.

Использование дуальных чисел в качестве аргумента кубического сплайна позволяет быстро получить значение первой производной кубического сплайна в силу следующего свойства:

$$f(a + b\epsilon) = f(a) + b\epsilon f'(a) \tag{9}$$

То есть, при вычислении значения функции в точке $a + b\epsilon$, дуальная часть полученного числа будет численно равна значению первой производной функции в точке a, если $b = 1$.

Листинг 7. Функция расчета первой производной

```
def calculate_first_derivative(sparse_set: np.ndarray,
                               x_coeffs: np.ndarray, y_coeffs: np.ndarray, factor: np.int32) \
        -> Tuple[List[ADN], List[ADN]]:
    x_der, y_der = [], []
    for index in range(1, len(sparse_set)-1):
        for t in np.arange(sparse_set[index], sparse_set[index+1], factor):
            t_dual = ADN(t, 1)
            t_j_dual = ADN(sparse_set[index], 0)
            x_der.append(spline(x_coeffs[0][index], x_coeffs[1][index],
                                x_coeffs[2][index], x_coeffs[3][index],
                                t_dual, t_j_dual))
            y_der.append(spline(y_coeffs[0][index], y_coeffs[1][index],
                                y_coeffs[2][index], y_coeffs[3][index],
                                t_dual, t_j_dual))
    return x_der, y_der
```

Функция `calculate_first_derivative` принимает на вход разреженное множество узлов, коэффициенты кубического сплайна для x и y, а также значение параметра M, задающего шаг перебора точек. Результатом работы являются два списка, содержащих значения производной для x и y в каждой точке.

13 Разработка функции построения нормали к заданному вектору

Для вычисления вектора нормали $R(t_j)$ к вектору $G(t_j)$ воспользуемся следующим соотношением:

$$(R_x(t_j), R_y(t_j) = (-G_y(t_j), G_x(t_j)) \tag{10}$$

Реализация функции построения нормали к заданному вектору приведена на листинге 8.

Листинг 8. Функция расчета первой производной

```
def normalize(x: np.ndarray, y: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
    x_normal, y_normal = -y, x
    return x_normal, y_normal
```

Функция `normalize` принимает на вход координаты конца вектора (x, y) и возвращает координаты конца нормали $(x_{\text{normal}}, y_{\text{normal}})$.

14 Построение векторов производных и нормалей

Для построения векторов производных $G(t_j)$ и нормалей $R(t_j)$ в соответствующих точках сплайна воспользуемся методом `plt.arrow` из библиотеки `matplotlib` и отобразим их на графике (рис. 6). Частота прореживания и масштаб выбраны таковыми, чтобы построенные вектора были визуально различимы на графике.

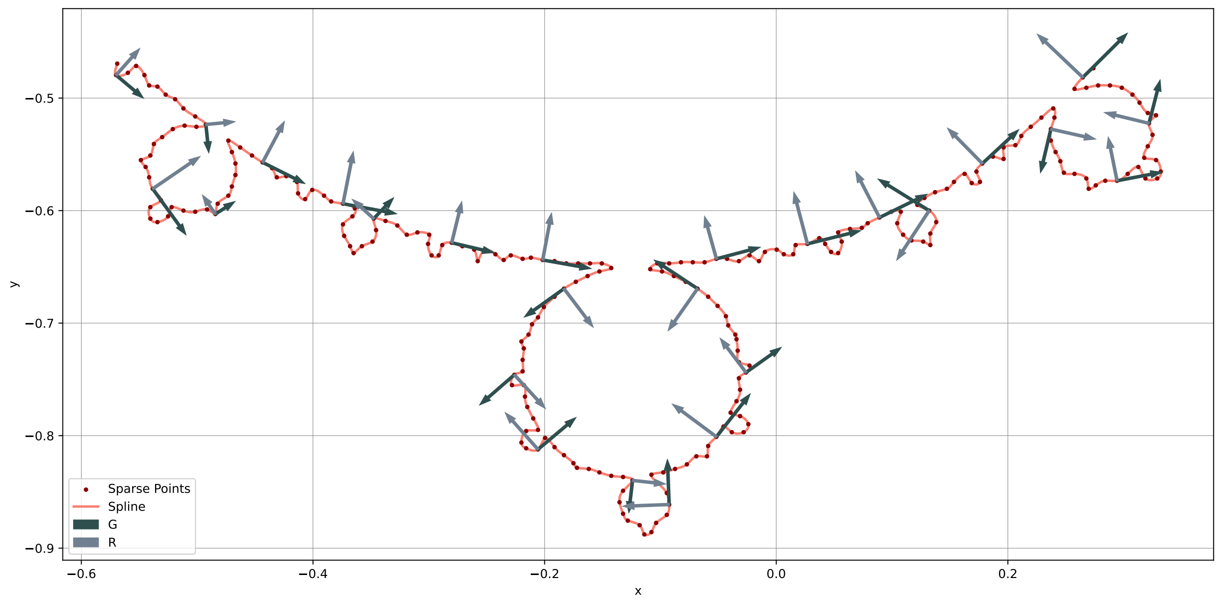

Рис. 6. Вектора нормалей и производных

Листинг 9. Построение векторов

```python
def display_vectors(x: np.ndarray, y: np.ndarray,
                    dx: np.ndarray, dy: np.ndarray,
                    step: np.int32, scale: np.float64,
                    settings: dict, label: str) -> None:
    for i in np.arange(0, len(x), step):
        if i == 0:
            plt.arrow(x[i], y[i], dx[i] * scale, dy[i]
                      * scale, label=label, **settings)
        else:
            plt.arrow(x[i], y[i], dx[i] * scale, dy[i] * scale, **settings)
```

Функция `display_vectors` (листинг 9) принимает на вход координаты начальных точек вектора, массивы изменений координат векторов, шаг для отображения и словарь для настроек отображения.

15 Заключение

В ходе выполнения лабораторной работы были достигнуты ключевые цели и выполнены все задачи кроме опциональной. Основные выводы следующие:

1. Интерполяция параметрическими кубическими сплайнами позволяет достичь достаточно высокой точности аппроксимации исходного множества точек, особенно при правильном выборе разреженного набора узлов.

2. Применение автоматического дифференцирования с использованием дуальных чисел обеспечивает вычислительную эффективность и точность при определении производных кубического сплайна.

Список использованных источников

1. Першин А.Ю. Лекции по курсу «Вычислительная математика». Москва, 2021. С. 140. URL: https://archrk6.bmstu.ru/index.php/f/810046.

2. Соколов, А.П. Инструкция по выполнению лабораторных работ (общая). Москва: Соколов, А.П., 2021. С. 9. URL: https://archrk6.bmstu.ru. (облачный сервис кафедры РК6).

3. Першин А.Ю., Соколов А.П., Гудым А.В. Сборник постановок задач на лабораторные работы по курсу «Вычислительная математика»: Учебное пособие. [Электронный ресурс]. Москва, 2023. С. 47. URL: https://archrk6.bmstu.ru. (облачный сервис кафедры РК6).

Выходные данные

Гавриш А.А.Отчет о выполнении лабораторной работы по дисциплине «Вычислительная математика». [Электронный ресурс] — Москва: 2023. — 18 с. URL: https://sa2systems.ru:88 (система контроля версий кафедры РК6)

2023, осенний семестр