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:	Гапеев Дмитрий Андреевич
Группа:	РК6-51Б
Тип задания:	лабораторная работа
Тема:	Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование

Студент	____________ подпись, дата	Гапеев Д.А. Фамилия, И.О.
Преподаватель	____________ подпись, дата	____________ Фамилия, И.О.

Москва, 2023

Содержание

Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование

Задание

Требуется (базовая часть):

1. Используя заранее подготовленный скрипт 1, выбрать произвольную область множества Мандельброта и построить фрагмент его границы (контура), сформировав файл contours.txt.
   Файл contours.txt содержит упорядоченную последовательность точек на плоскости $P = \{(x_i, y_j)\}_{i=1}^{N}$, принадлежащих выбранному фрагменту границы фрактала **c**. Сопоставляя каждой паре координат естественную координату t, предполагать, что $x_i = x(t_i), y_i = y(t_i)$.

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

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

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

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

5. Вычислить расстояния $\rho[(\widetilde{x}(t_i), \widetilde{y}(t_i)), (x(t_i), y(t_i))]$ 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 размером $\hat{N}-1$ строк на 8 столбцов. Функция lab1_base должна реализовывать базовую часть задания.

Требуется (продвинутая часть):

8. Используя концепцию дуальных чисел $v = a + \varepsilon b$, $e^2 = 0$, 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 $G(t) = \frac{d}{dt}(\widetilde{x}(t), \widetilde{y}(t))$.

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.

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы - знакомство с такими математическими библиотеками языка программирования Python, как matplotlib, numpy. Реализация с использованием указанных библиотек локальной интерполяции кубическими сплайнами для фрагмента фрактала **c**, а также автоматического дифференцирования при помощи дуальных чисел.

1 Интерполяция параметрическими кубическими сплайнами

1.1 Генерация контура множества Мандельброта

Используя скрипт для генерации точек, был выбран фрагмент контура множества Мандельброта в произвольной области плоскости. В результате выполнения скрипта создан файл contours.txt, содержащий упорядоченную последовательность точек на плоскости $P = \{(x_i, y_i)\}_{i=1}^{N}$, принадлежащих выбранному фрагменту границы. Каждой паре точек сопоставлена естественная координата t, так что $x_i = x(t_i), y_i = y(t_i)$. Полученный контур содержит 2403 точки. При последующем решении задач предполагается, что t принимает значения в промежутке $[0, 2403]$, то есть t_i - множество индексов точек.

1.2 Загрузка и визуализации множества точек P

Для загрузки множества точек P использована функция *numpy.loadtxt*. Затем выполнено разделение координат точек по массивам x и y. Для визуализации разработана функция **draw_points(x, y)** , которая принимает в качестве аргументов множества x и y и изображает множество точек P при помощи функций библиотеки matplotlib (рис. 1.).

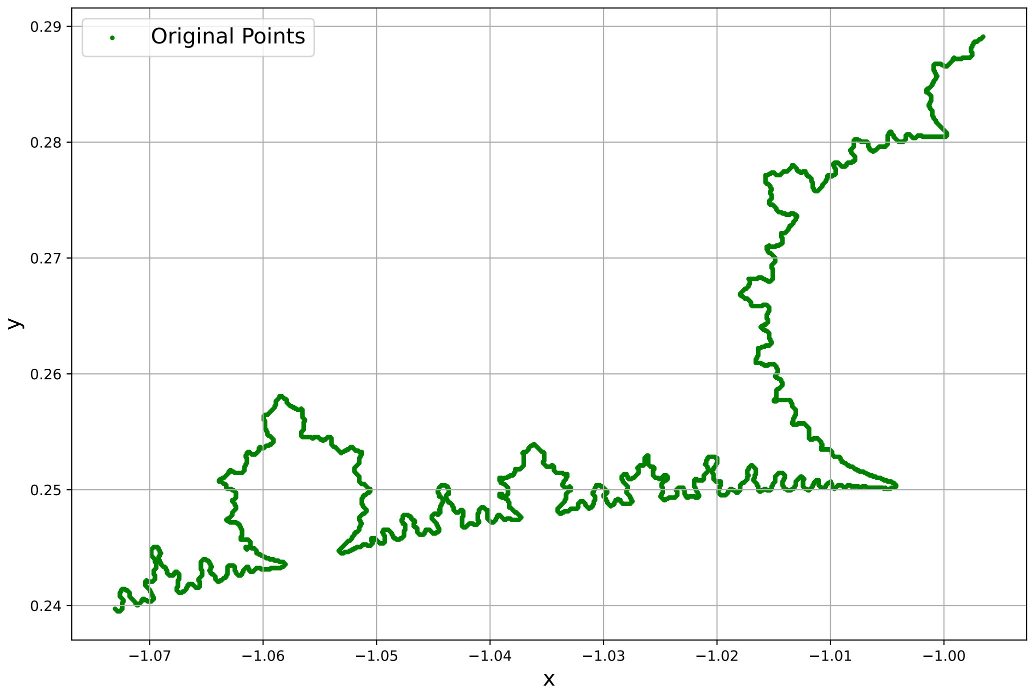

Рис. 1. Визуализация множества точек P

1.3 Определение подмножества $\hat{P}$ для интерполяции

Разреженное множество задается как множество так: $\hat{P} = \{(x_j, y_j)\}_{j=1}^{\hat{N}}, \hat{N} = \lfloor N/M \rfloor, j = M \times i, \hat{P} \subset P$. Полагается, что $M = 10$. Для создания разреженного множества интерполяционных узлов $\hat{P}$ сформирован отдельный массив индексов *sparseset* в диапозоне $[0, N]$ с шагом $M = 10$. Сформированы массивы разреженных множеств для каждой из координат: $sparseX$ и $sparseY$, при помощи слайсинга к исходному множеству точек контура P.

1.4 Нахождение коэффициентов параметрического кубического сплайна

По определению, если функция $f(x)$ задана в n интерполяционных узлах $a = x_1, x_2, ..., x_n = b$ на отрезке $[a; b]$, то кубическим сплайном для функции $f(x)$ называется функция $S(x)$, имеющая вид:

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

где $a_i,\ b_i,\ c_i,\ d_i$ - коэффициенты кубического сплайна.
Кубический сплайн $S_j(t)$, действующий на интервале $[t_j, t_{j+1}]$, является слагаемым для вычисления естественного параметрического сплайна (1):

$$S_j(t) = a_j + b_j(t - t_j) + c_j(t - t_j)^2 + d_j(t - t_j)^3 \tag{2}$$

Необходимым является нахождение $8(\hat{N} - 1)$ коэффициентов.
Матричное уравнение для коэффициентов имеет вид[1]:

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ h_1 & 2(h_2 + h_1) & h_2 & 0 & \cdots & 0 \\ 0 & h_2 & 2(h_3 + h_2) & h_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{\hat{N}-2} & 2(h_{\hat{N}-2} + h_{\hat{N}-1}) & h_{\hat{N}-1} \\ 0 & \cdots & \cdots & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{\hat{N}-1} \\ c_{\hat{N}} \end{bmatrix} =$$

$$= \begin{bmatrix} 0 \\ \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1) \\ \frac{3}{h_3}(a_4 - a_3) - \frac{3}{h_2}(a_3 - a_2) \\ \vdots \\ \frac{3}{h_{\hat{N}-1}}(a_{\hat{N}} - a_{\hat{N}-1}) - \frac{3}{h_{\hat{N}-2}}(a_{\hat{N}-1} - a_{\hat{N}-2}) \\ 0 \end{bmatrix}, \tag{3}$$

где $h_j = t_{j+1} - t_j = M = 10$, поскольку множество t_j состоит из индексов каждой 10-ой точки множества P. Для разработки функции вычисления коэффициентов кубического сплайна **calculate_spline_coeffs(sparse, factor)** (Листинг 1) необходимо на основе интерполяционных узлов $sparseX, sparceY$ заполнить элементы матричного уравнения (3). Аргумент $factor$ - значение параметра M. Алгоритм ее заполнения следующий:

1. Создать нулевую матрицу размером $\hat{N} \times \hat{N}$, где $\hat{N}$ - количество интерполяционных узлов. Данная матрица будет заполнена в соответствии с первой матрицей из (3).

2. Создать нулевой вектор размером $\hat{N}$. Данный вектор будет результирующим вектором из (3). Для создания нулевого вектора использована функция numpy.zeros;

3. Заполнить элементы нулевой матрицы $[1, 1]$ и $[\hat{N}, \hat{N}]$ единицами в соответствии с (3);

4. В цикле продолжить заполнение элементов строк $i = 2, ..., \hat{N} - 1$ в соответствии со следующим алгоритмом: со смещением заполняются элементы по бокам от главной диагонали в соответствии с (3), затем заполняется элемент главной диагонали, являющийся удвоенной суммой боковых элементов, каждую итерацию смещение трех элементов увеличивается на 1.;

5. В цикле заполнить элемент результирующего вектора, соответствующий данной итерации в соответствии с формулами из (3). Элемент на каждой итерации имеет вид $(3/h_j) * (sparse[i+1] - sparse[i]) - (3/h_j) * (sparse[i] - sparse[i-1])$

В итоге, получена заполненная матрица и результирующий вектор. Необходимо решить матричное уравнение типа

$$AX = B, \tag{4}$$

где A - заполненная матрица, X - вектор-столбец коэффициентов c_j, B - результирующий вектор. Для решения матричного уравнения (4) использована функция np.linalg.solve(). Таким образом найден коэффициент c_j из вектора коэффициентов. Коэффициенты a_j, b_j, d_j имеют следующий вид[1]:

$$a_j = S(t_j)$$

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(c_{j+1} + 2c_j)$$

$$d_j = \frac{c_{j+1} - c_j}{3h_j}$$

Листинг 1. Функция calculate_spline_coeffs(sparse, factor)

```
def calculate_spline_coeffs(sparse, factor):
    n = len(sparse)
    A = np.zeros((n, n))
    A[0, 0] = A[-1, -1] = 1
    for i in range(1, n - 1):
        A[i, i - 1] = factor
        A[i, i] = 4 * factor
        A[i, i + 1] = factor
    B_matrix = np.zeros(n)
    for i in range(1, n - 1):
        B_matrix[i] = (3 / factor) * (sparse[i+1] - sparse[i]) \
                      - (3 / factor) * (sparse[i] - sparse[i-1])
    c = np.linalg.solve(A, B_matrix)
    a = sparse[:-1]
    d = [(c[i + 1] - c[i]) / (3 * factor) for i in range(n - 1)]
    b = [(1 / factor) * (sparse[i + 1] - sparse[i])
         - (factor / 3) * (c[i + 1] + 2*c[i]) for i in range(n - 1)]
    return a, b, c[:-1], d
```

1.5 Вычисление расстояний, среднего и стандартного отклонения

Для вычисления евклидовых расстояний на плоскости определена функция **euclidean_dist(x1, y1, x2, y2)**, аргументами которой являются координаты двух точек. Функция **euclidean_dist(x1, y1, x2, y2)** возвращает расстояние между двумя точками, то есть $\sqrt{(x_1^2 - x_2^2) + (y_1^2 - y_2^2)}$

Также для расчета значения кубического сплайна в точке t_i по формуле (2) изначально требуется определить, к какому отрезку $[j; j+1], j = 0, ..., \hat{N} - 1$ принадлежит t_i. Это необходимо для подстановки в формулу (1) коэффициентов кубического сплайна. Определение принадлежности точки к отрезку выполняется в цикле, используя следующее сравнение: t_i целочисленно делится на M и если полученное значение меньше количества сплайнов то переменная t_i запоминается. Полученное значение при целочисленном делении будет являться индексом необходимых a_{jk}(или b_{jk}). В случае если рузльтат целочисленного деления не меньше чем количество сплайнов, то результат целочисленного деления уменьшается. Найденные значения, а также коэффициенты кубического сплайна подставляются в (2)

Таким образом находятся расстояния для каждой пары точек $(x(t_i), y(t_i))$ и $(\widetilde{x}(t_i), \widetilde{y}(t_i))$ при помощи функции **calculate_distances(sparseset, a_j, b_j, x, y, factor)** (Листинг 2).

Листинг 2. Функция **calculate_distances(sparseset, a_j, b_j, x, y, factor)**

```
def calculate_distances(sparseset, aj, bj, x, y, factor):
    distances = []
    for t in range(len(x)):
        i = int(t // factor)
        if i >= len(sparseset) - 1:
            i = len(sparseset) - 2
        t_j = sparseset[i]
        x_spline = spline(i, aj, t, t_j)
        y_spline = spline(i, bj, t, t_j)
        dist = euclidean_dist(x_spline, y_spline, x[t], y[t])
        distances.append(dist)
    return distances
```

Расчет среднего и стандартного отклонения реализуется при помощи функций **np.mean()** и **np.std()** Значения среднего и стадартного отклонения: $\mu(X) \approx 0.000126284$, $\sigma(X) \approx 0.0000966528$.

1.6 Отображение полученного сплайна вместе с исходным множеством точек P, а также узловыми точками $\hat{P}$.

Для отображения сплайна необходимо вычислить значения сплайна в точке t согласно формуле (2) первоначально требуется определить к какому отрезку $[j, j+1], j = 0, ..., \hat{N} - 1$ принадлежит t (аналогично п.1.5).

Реализована функция **generate_spline_points(sparseset, aj, bj, h, t_N, factor)**

(Листинг 3) для вычисления *xval* и *yval* при помощи формулы (2), где t изменяется в интервале $[0, t_N]$ с шагом h=0.1. Отображен полученный сплайн, исходное множество точек P, разреженное множество узловых точек $\hat{P}$ (рис. 2).
Наблюдаемая ошибка ошибка интерполяции связана с малым количеством узлов для интерполяции, а также ввиду сложной формой множества Мандельброта. Для уменьшении ошибки можно увеличить количество узлов интерполяции, то есть выбрать $M < 10$. Также можно попробовать применение методов сглаживания, которое улучшить качество интерполяции.

Листинг 3. Функция **generate_spline_points(sparseset, aj, bj, h, t_N, factor)**

```python
def generate_spline_points(sparseset, aj, bj, h, t_N, factor):
    xval, yval = [], []
    for t in np.arange(0, t_N, h):
        i = int(t // factor)
        if i >= len(sparseset) - 1:
            i = len(sparseset) - 2
        t_j = sparseset[i]
        x_approx = spline(i, aj, t, t_j)
        y_approx = spline(i, bj, t, t_j)
        xval.append(x_approx)
        yval.append(y_approx)
    return xval, yval
```

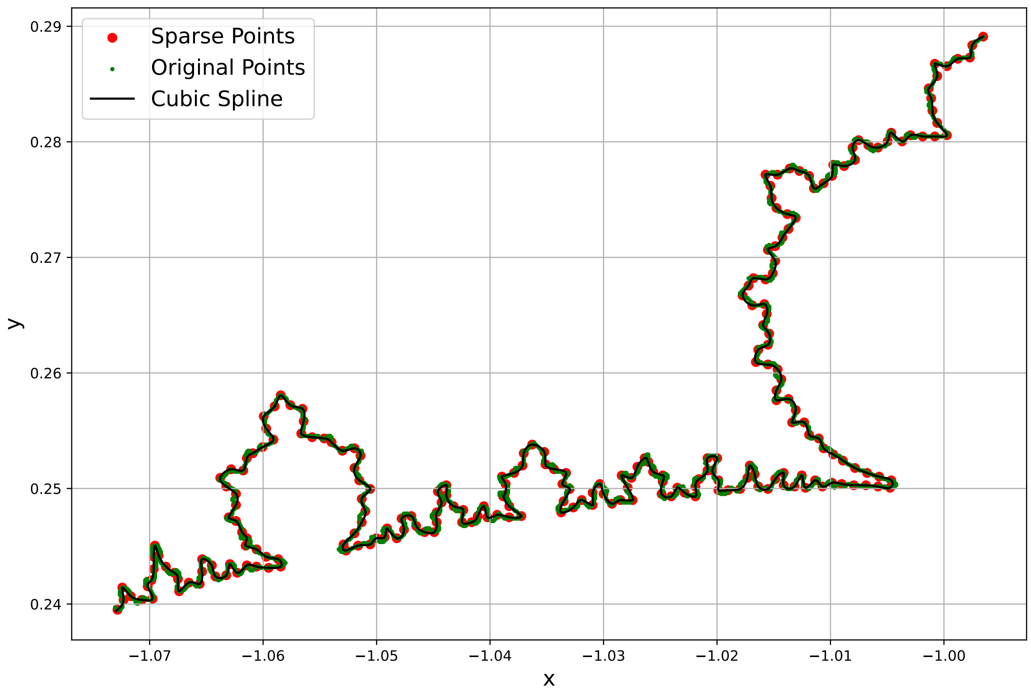

Рис. 2. Визуализация полученного сплайна, исходного множества точек P, множества узловых точек $\hat{P}$

1.7 Реализация функции для выполнения базовой части задания.

Реализована функция **lab1_base(filename_in:str, factor:int, filename_out:str)** (Листинг 4), в которой выполнен ряд операций. Первоначально, из файла, указанного в параметре 'filename_in', загружается упорядоченная последовательность точек P. Далее, с использованием параметра 'factor', формируется разреженное множество интерполяционных узлов $\hat{P}$.

Для каждого измерения x и y решается СЛАУ для нахождения коэффициентов a_{jk} и b_{jk} естественного параметрического кубического сплайна. Полученные коэффициенты сохраняются в виде матрицы в файл, имя которого задано параметром 'filename_out'.

Вычисляются расстояния $\rho[(\widetilde{x}(t_i), \widetilde{y}(t_i)), (x(t_i), y(t_i))]$ между аппроксимированными и исходными точками контура. Результаты представлены в виде среднего значения и стандартного отклонения.

Визуализация исходного множества точек, разреженного множества и полученного сплайна производится на одном графике. Все задания базовой части инкапсулированы внутрь реализованной функции.

Листинг 4. Функция **lab1_base(filename_in:str, factor:int, filename_out:str)**

```
def lab1_base(filename_in:str, factor:int, filename_out:str):
    P = np.loadtxt(filename_in)
    x, y = P[:, 0], P[:, 1]
    sparseset = np.arange(0, len(P), factor)
    sparseX, sparseY = x[sparseset], y[sparseset]
    a_x, b_x, c_x, d_x = calculate_spline_coefficients(sparseset, sparseX)
    a_y, b_y, c_y, d_y = calculate_spline_coefficients(sparseset, sparseY)
    aj = np.column_stack((a_x, b_x, c_x, d_x))
    bj = np.column_stack((a_y, b_y, c_y, d_y))
    t_N = len(P) - 1
    h = 0.1
    xval, yval = generate_spline_points(sparseset, aj, bj, h, t_N)
    distances = calculate_distances(sparseset, aj, bj, x, y)
    mean_distance = np.mean(distances)
    std_distance = np.std(distances)
    draw_spline_and_points(sparseX, sparseY, x, y, xval, yval)
    print(f"Mean Distance: {mean_distance}, Standard Deviation: {std_distance}")
    coefficients_matrix = np.column_stack((aj, bj))
    np.savetxt(filename_out, coefficients_matrix, fmt='%.8f')
```

2 Автоматическое дифференцирование при помощи дуальных чисел

2.1 Разработка класса AutoDiffNum

Автоматическое вычисление производных может быть осуществлено через применение дуальных чисел. Арифметические операции с дуальными числами[1]:

$$\langle a, b\rangle + \langle c, d\rangle = \langle a + c, b + d\rangle \tag{5}$$

$$\langle a, b\rangle - \langle c, d\rangle = \langle a - c, b - d\rangle \tag{6}$$

$$\langle a, b\rangle \times \langle c, d\rangle = \langle a \cdot c, b \cdot c + a \cdot d\rangle \tag{7}$$

$$\langle a, b\rangle^n = \langle a^n, b \cdot n \cdot a^{n-1}\rangle \tag{8}$$

Разработан класс AutoDiffNum, который использует логику дуальных чисел и перегрузку математических операций для автоматического вычисления производной некоторой функции. Объект этого класса принимает два параметра при инициализации: реальную и дуальную части.

В конструкторе **__init__** класса AutoDiffNum инициализируются два атрибута: self.a и self.b. В классе self.a представляет собой реальную часть дуального числа, а self.b — его дуальную часть. Также перегружены операторы **__add__** и **__sub__** в функции **class AutoDiffNum** (Листинг 5) для сложения(5) и вычитания(6) дуальных чисел.

Листинг 5. **class AutoDiffNum**(первая часть)

```
class AutoDiffNum:
    def __init__(self, a, b):
        self.a = a # real path
        self.b = b # dual path
    def __add__(self, other):
        if isinstance(other, AutoDiffNum):
            return AutoDiffNum(self.a + other.a, self.b + other.b)
        else:
            return AutoDiffNum(self.a + other, self.b)
    def __sub__(self, other):
        if isinstance(other, AutoDiffNum):
            return AutoDiffNum(self.a - other.a, self.b - other.b)
        else:
            return AutoDiffNum(self.a - other, self.b)
```

Далее реализуются перегрузка операторов **__mul__** и **__power__** в функции **class AutoDiffNum** (Листинг 6), которые соответствуют умножению(7) и возведению в степень(8). Помимо прочего, присутствует конструктор **__repr__** для строкового представления дуального числа.

Листинг 6. **class AutoDiffNum**(вторая часть)

```
    def __mul__(self, other):
        if isinstance(other, AutoDiffNum):
            return AutoDiffNum(self.a * other.a, self.a * other.b + self.b * other.a)
        else:
            return AutoDiffNum(self.a * other, self.b * other)

    def __pow__(self, power):
        if isinstance(power, AutoDiffNum):
            raise ValueError("Power should be a real number.")
        else:
            return AutoDiffNum(self.a ** power, self.b * power * self.a ** (power - 1))

    def __repr__(self):
        return f"{self.a} + {self.b}e"
```

2.2 Реализация функции автоматического расчета первой производной

При программной реализации использовано важное свойство дуальных чисел[1]:

$$f(a + b\epsilon) = f(a) + b\epsilon f'(a)$$

Значит при вычислении значения функции в точке $a + b\epsilon$, дуальная часть полученного числа равна значению первой производной функции в точке a, если $b = 1$. Для автоматического вычисления первой производной разработана функция

spline_derivative(i, aj, t, t_j) (Листинг 7), которая вычисляет значение производной $G(t)$ в точке t на отрезке $[t_j, t_{j+1}]$ с использованием дуальных чисел. Для этого создано дуальное число $t+1\epsilon$, которое подставляется в кубический сплайн. При помощи перегрузки арифмитических операторов вычислено значение $G(t+1\epsilon)$. Производная $G(t)$ содержится в дуальной части $G(t+1\epsilon)$.

Листинг 7. Функция **spline_derivative(i, aj, t, t_j)**

```
    def spline_derivative(i, aj, t, t_j):
    # Инициализация дуального числа для точки t
    t_dual = AutoDiffNum(t, 1)
    # Вычисление производной кубического сплайна в точке t
    G_t = (t_dual - t_j) * aj[i][1] + (t_dual - t_j) ** 2 * aj[i][2] + (t_dual - t_j) ** 3
        * aj[i][3] + aj[i][0]
    return G_t
```

2.3 Реализация функции построения нормали к заданному вектору

Вектор $G(t_j)$ на каждом узле t_j кубического сплайна определяется как $G(t_j) = (G_x(t_j), G_y(t_j))$, где $G_x(t_j)$ и $G_y(t_j)$ - производные функции кубического сплайна по x и y. Ввиду того что, нормаль $R(t_j)$ перпендикулярна к вектору $G(t_j)$, то

$$R(t_j) = (-G_y(t_j), G_x(t_j)) \tag{9}$$

Реализована функция **calculate_tangent_vectors(sparseset, aj, bj, t_N, factor)** (Листинг 8), которая заполняет два массива $G\_t\_values\_x$ и $G\_t\_values\_y$ компонентами x и y векторов касательных $G(t_j)$ для каждого узла t_j. Компоненты $G_{t,x}$ и $G_{t,y}$ вычисляются при помощи функции $spline\_deritave$, которая использует автоматическое дифференцирование с помощью класса $AutoDiffNum$ для вычисления производных. Реализована функция **calculate_normals(G_t_values_x, G_t_values_y)** (Листинг 9), которая вычисляет нормали $R(t_j)$ при помощи формулы (9)

Листинг 8. Функция **calculate_tangent_vectors(sparseset, aj, bj, t_N, factor)**

```
def calculate_tangent_vectors(sparseset, aj, bj, t_N, factor):
    G_t_values_x = []
    G_t_values_y = []
    for t in np.arange(0, t_N, factor):
        i = int(t // factor)
        if i >= len(sparseset) - 1:
            i = len(sparseset) - 2
        t_j = AutoDiffNum(sparseset[i], 0)
        G_t_x = spline_derivative(i, aj, t, t_j)
        G_t_y = spline_derivative(i, bj, t, t_j)
        G_t_values_x.append(G_t_x.b)
        G_t_values_y.append(G_t_y.b)
    return G_t_values_x, G_t_values_y
```

Листинг 9. Функция **calculate_normals(G_t_values_x, G_t_values_y)**

```python
def calculate_normals(G_t_values_x, G_t_values_y):
    R_t_values_x = []
    R_t_values_y = []
    for i in range(len(G_t_values_x)):
        R_t_values_x.append(-G_t_values_y[i])
        R_t_values_y.append(G_t_values_x[i])
    return R_t_values_x, R_t_values_y
```

2.3 Реализация функции построения касательной и нормали

Для визуализации использованы стрелки, которые начинаются в узлах t_j и направлены вдоль векторов $G(t_j)$ и $R(t_j)$. Визуализация стрелок осуществляется при помощи функции $plt.arrow()$. Частота прореживания выбрана таковой, чтобы на контуре было не менее 5 точек. Масштаб стрелок выбран так, чтобы их было легко различить на графике(рис. 3.).

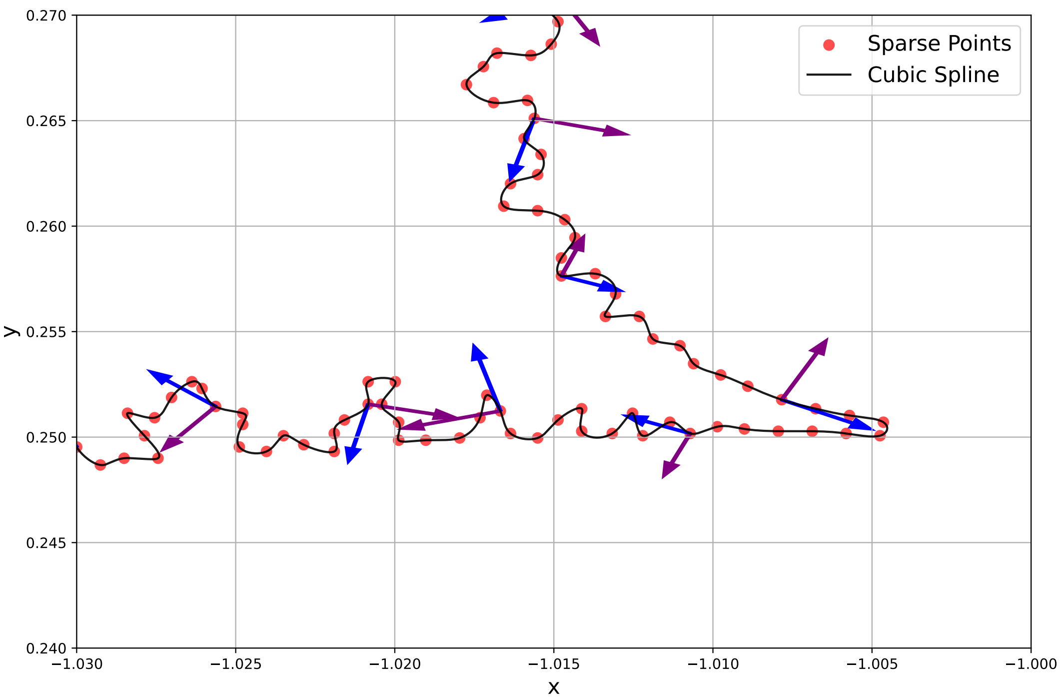

Рис. 3. Визуализация векторов касательных $G(t_j)$(синий цвет) и векторов нормали $R(t_j)$(фиолетовый цвет)

Заключение

1. *Успешно осуществленно знакомство с математическими библиотеками Python - matplotlib и numpy.*

2. *С применением указанных библиотек была реализована локальная интерполяция кубическими сплайнами. Целевым объектов для интерполяции послужил контур множества Мандельброта.*

3. *Реализован механизм автоматического дифференцирования на основе дуальных чисел.*

По итогу, изучены интерполяция кубическими сплайнами, автоматическое дифференцирование с использованием дуальных чисел, и проанализированы результаты их работы.

Список использованных источников

1. Першин А.Ю. Лекции по курсу «Вычислительная математика». Москва, 2018-2021. 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., 2018-2021. 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: hhttps://arch.rk6.bmstu.ru.

Выходные данные

Гапеев Д.А.Отчет о выполнении лабораторной работы по дисциплине «Вычислительная математика». [Электронный ресурс] — Москва: 2023. — 15 с. URL: https://sa2systems.ru:88 (система контроля версий кафедры РК6)

Постановка: *аспирант кафедры РК-6, Гудым А.В.*

Решение и вёрстка: *студент группы РК6-51Б, Гапеев Д.А.*

2023, осенний семестр