Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего 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	Шпиталев Даниил Олегович
Группа	РК6-56Б
Тип задания	лабораторная работа 1 Вариант 6
Тема лабораторной работы	Интерполяция параметрическими кубическими сплайнами и автоматическое дифференцирование

Студент _____ **Шпиталев Д.О.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Соколов А.П.**
подпись, дата *фамилия, и.о.*

Москва, 2023 г.

Оглавление

Задание на лабораторную работу

Требуется (базовая часть): 1. Используя заранее подготовленный скрипт 1, выбрать произвольную область множества Мандельброта и построить фрагмент его границы (контура), сформировав 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 = $\{(x_i , y_i)\}_i^N$, принадлежащих выбранному фрагменту границы фрактала с. 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 $\hat{P} = \{(x_i , y_i)\}_j^{\hat{N}}$, $\hat{N} = \lfloor N/M \rfloor$, $j = M \times i$, $\hat{P} \subset P$. 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:

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

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

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

6. Вычислить расстояния $\rho\,[(\tilde{x}(t_i), \tilde{y}(t_i)), (x(t_i), y(t_i))]$ и представить вывод 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 $\widehat{N} - 1$ строк на 8 столбцов. Функция lab1_base должна реализовывать базовую часть задания.

Требуется (продвинутая часть):

8. Используя концепцию дуальных чисел $v = a + \varepsilon b$, $\varepsilon^2 = 0$, 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.

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).

Цель выполнения лабораторной работы

Освоить такой метод кусочной интерполяции, как построение кубических сплайнов, аппроксимировать множество точек контура Мандельброта, построив гладкую кривую, проходящую через некоторые точки данного множества, построить касательные и нормали к полученной кривой, изучив и использовав метод автоматического дифференцирования, основанный на концепции дуальных чисел.

Базовая часть.

Пункт 1.

Подготовка файла с точками контура Мандельброта

Запустим заранее подготовленный скрипт, визуализирующий множество Мандельброта. Получим уникальный фрагмент его границы (изображен на рис. 1), сохраним файл contours.txt со всеми точками этого фрагмента.

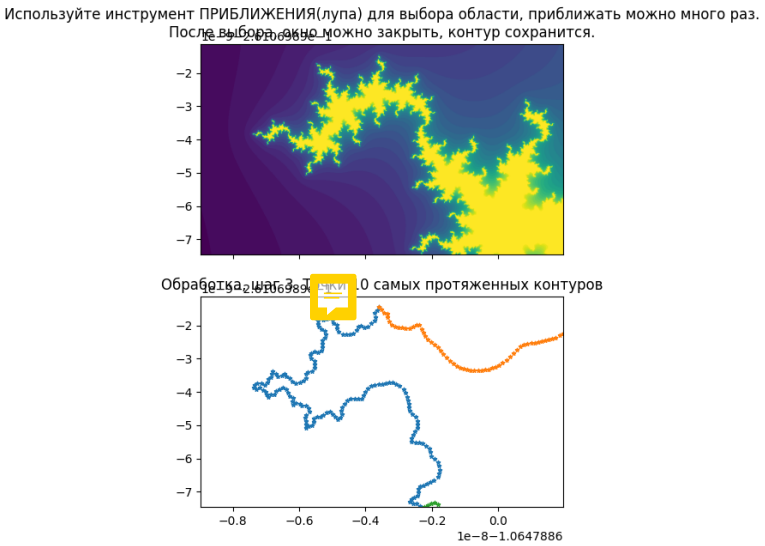

Рис. 1: полученный контур множества Мандельброта

Пункт 2.

Разработка кода для загрузки и визуализации множества точек из файла contours.txt

Функция, выполняющая пункт 2 представлена в листинге 1. Функция plot_mandelbrot_points(filename) считывает данные из файла и извлекает координаты x и y точек. Затем она использует функцию plt.scatter() из библиотеки matplotlib.pyplot для визуализации точек. В конце функция возвращает извлеченные координаты в виде списка списков, где первый список содержит координаты x, а второй список содержит координаты y. Результат работы представлен на рис. 4.

Листинг 1: функция для отображения множества точек из файла

```
def plot_mandelbrot_points(filename):
    mandelbrot_points = [[], []]
    with open(filename, 'r') as file:
```

```
        for line in file:
            columns = line.split()
            if len(columns) > 1:
                mandelbrot_points[0].append(float(columns[0]))
                mandelbrot_points[1].append(float(columns[1]))
        plt.scatter(mandelbrot_points[0], mandelbrot_points[1], 0.5, 'purple' )
        plt.title('множество точек Мандельброта')
        return mandelbrot_points
```

Пункт 3.

Задание разреженного множества интерполяционных узлов

Функция, выполняющая пункт 3 представлена в листинге 2. Аргумент M отвечает за разреженность, чем больше его значение, тем более разреженным становится массив, что приводит к меньшему количеству интерполяционных узлов и снижению точности интерполяции. Каждой координате x и y, удовлетворяющей условие разреженности (каждая M-ая точка), в соответствие ставится параметр t, являющийся целым числом. Каждую итерацию t увеличивается на 1.

Листинг 2: функция создания разреженного множества интерполяционных узлов

```
def create_sparse_matrix(filename, M): #построение разреженного множества
интерполяционных узлов t, x, y
    sparse_matrix = [[], [], []]       # t целые числа с шагом 1
    t = 0
    j = 0
    with open(filename, 'r') as file:
        for line in file:
            columns = line.split()
            if len(columns) > 1 and j % M == 0:
                sparse_matrix[0].append(int(t))
                sparse_matrix[1].append(float(columns[0]))
                sparse_matrix[2].append(float(columns[1]))
                t+=1
            j+=1
    return sparse_matrix
```

Пункт 4.

Нахождение коэффициентов естественного параметрического кубического сплайна

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \quad (1)$$

В формуле (1) представлен вид кубического сплайна, a_j, b_j, c_j, d_j – коэффициенты кубического сплайна на j интервале.

Чтобы найти коэффициенты естественного параметрического кубического сплайна на каждом интервале, нужно решить матричное уравнение (3).

$$Ac = b \tag{2}$$

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

Результатом решения будут коэффициенты c_j, из которых потом можно выразить остальные коэффициенты по формулам на рис. 2.

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

Рис. 2: формулы для вывода коэффициентов сплайна, через коэффициент c

Для решения матричного уравнения (3), нужно заполнить матрицы A (функция `get_matrix_A(sparse_matrix)`) и b (функция `get_matrix_b(sparse_matrix)`). Затем нужно найти матрицу c, при помощи метода Гаусса (функция `gauss_method(A, b)`). Воспользовавшись формулами (рис. 2), найти остальные коэффициенты (функция `get_coefs_spline(sparse_matrix, c)`). Фрагмент кода, выполняющий данную последовательность действий приведен в листинге 3.

Листинг 3: фрагмент кода, выполняющий нахождение разрешающих коэффициентов

```
def get_matrix_A(sparse_matrix):   #функция построения матрицы A, использующейся для нахождения коэффициентов через матричное уравнение
    n = len(sparse_matrix[0])
    matrix_A = np.zeros((n, n)) #функция принимает разреженную матрицу координат (x,t) или (y,t)
    matrix_A[0][0] = 1
    matrix_A[n-1][n-1] = 1
    for i in range(1, n-1):
        j = i - 1
        matrix_A[i][j] = int(sparse_matrix[0][j+1]-sparse_matrix[0][j])
        matrix_A[i][j+1] = 2 * (sparse_matrix[0][j+2]-sparse_matrix[0][j])
        matrix_A[i][j+2] = sparse_matrix[0][j+2]-sparse_matrix[0][j+1]
    return matrix_A

def get_matrix_b(sparse_matrix):
    n = len(sparse_matrix[0])
    matrix_b = np.zeros(n)
    for i in range(1, n-1):
        matrix_b[i] = 3 * (sparse_matrix[1][i+1] - sparse_matrix[1][i]) / (sparse_matrix[0][i+1] - sparse_matrix[0][i]) - 3 * (sparse_matrix[1][i] - sparse_matrix[1][i-1]) / (sparse_matrix[0][i] - sparse_matrix[0][i-1])
        matrix_b[0] = 0
        matrix_b[n-1] = 0
    return matrix_b
```

```
def get_coefs_spline(sparse_matrix, c): # a b c d коэффициенты кубического
сплайна
    n = len(c)
    coefs = np.zeros((n, 4))
    for i in range(0, n):
        coefs[i][0] = sparse_matrix[1][i] #коэффициент a
    for i in range(0,n-1):
        h_i = (sparse_matrix[0][i+1]-sparse_matrix[0][i])
        coefs[i][1] = (coefs[i+1][0]-coefs[i][0]) / h_i - h_i / 3 * (c[i+1] + 2
* c[i]) #коэффициент b
        coefs[i][2] = c[i] #коэффициент c
        coefs[i][3] = (c[i+1] - c[i]) / (3 * h_i) #коэффициент d
    return coefs[:-1]

def gauss_method(A, b): #метод Гаусса
    n = len(A)
    for i in range(n):
        max_row = i
        for j in range(i+1, n):
            if abs(A[j][i]) > abs(A[max_row][i]):
                max_row = j
        A[[i, max_row]] = A[[max_row, i]]
        b[[i, max_row]] = b[[max_row, i]]
        for j in range(i+1, n):
            ratio = A[j][i] / A[i][i]
            A[j] -= ratio * A[i]
            b[j] -= ratio * b[i]
    decision = np.zeros(n)
    for i in range(n-1, -1, -1):
        decision[i] = (b[i] - np.dot(A[i][i+1:], decision[i+1:])) / A[i][i]
    return decision

sparse_matrix = create_sparse_matrix(filename_in, factor)
    x = [sparse_matrix[0], sparse_matrix[1]]
    y = [sparse_matrix[0], sparse_matrix[2]]
    A_x = get_matrix_A(x)
    A_y = get_matrix_A(y)
    b_x = get_matrix_b(x)
    b_y = get_matrix_b(y)
    c_x = gauss_method(A_x, b_x)
    c_y = gauss_method(A_y, b_y)
    xcoefs = get_coefs_spline(x, c_x)
    ycoefs = get_coefs_spline(y, c_y)
```

Результатом работы фрагмента кода (листинг 3) станет получение 2 массивов размерности [n-1][4] коэффициентов a_j, b_j, c_j, d_j для параметрического кубического сплайна по измерению x и y (n – число интерполяционных узлов).

Пункт 5.

Отображение полученного сплайна

функция, представленная в листинге 4, выполняет пункт 5 задания. Данная функция получает на вход ранее найденные коэффициенты сплайнов. Каждому j интервалу интерполяции соответствуют свои 4 коэффициента сплайна по измерению x и 4 по измерению y, а также параметр t от j до j + 1. Зная уравнение

кубического сплайна на каждом интервале, подставив в него 10 значений параметра t из соответствующего интервала можно получить визуально гладкую кривую. По завершении работы функция возвращает лист листов координат, по которым были построены все сплайны.

Листинг 4: функция для отображения кубических сплайнов

```
def show_cube_splines(xcoefs, ycoefs, M):
    n = len(xcoefs)  # Получаем количество интервалов
    spline_coordinates = [[], []]
    for i in range(n):
        a = xcoefs[i][0] #извлекаем коэффициенты
        b = xcoefs[i][1]
        c = xcoefs[i][2]
        d = xcoefs[i][3]
        t = np.linspace(i, i+1, M)  #(для частоты шага h = 0.1 и M = 10 delta(t)
= 1) берем 10 точек из каждого интервала
        x = a + b * (t - i) + c * (t - i) ** 2 + d * (t - i) ** 3  # Вычисляем
значения кубического сплайна при некоторых t из интервала
        for j in range(0, M):
            spline_coordinates[0].append(x[j])
        a = ycoefs[i][0]
        b = ycoefs[i][1]
        c = ycoefs[i][2]
        d = ycoefs[i][3]
        y = a + b * (t - i) + c * (t - i) ** 2 + d * (t - i) ** 3
        for j in range(0, M):
            spline_coordinates[1].append(y[j])
```

Результат работы можно увидеть на рис. 3, 4

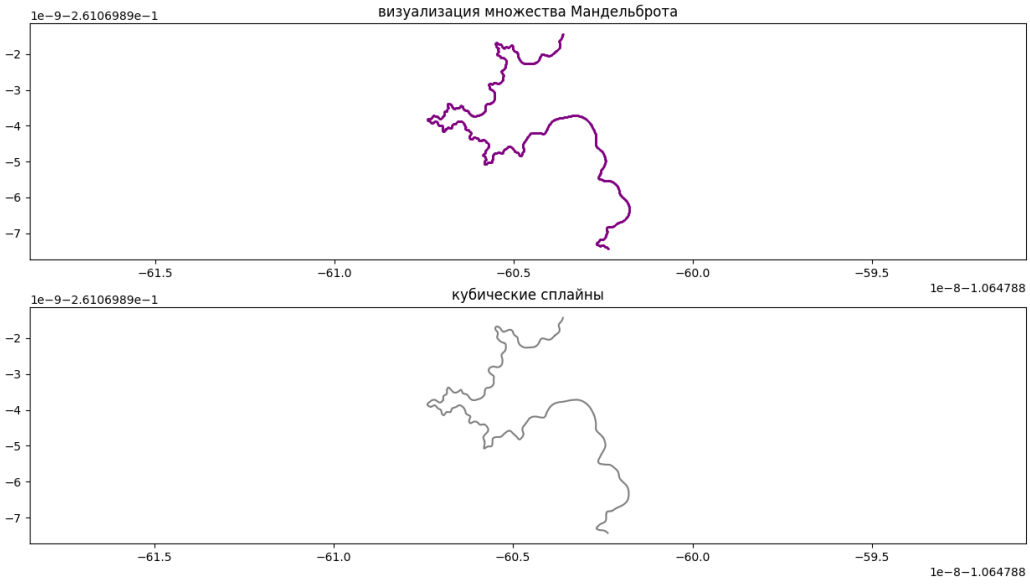

Рис. 3: множество точек контура Мандельброта (сверху) и кусочно-заданные сплайны (снизу)

Для уменьшения ошибки интерполяции, можно уменьшить M, что приведет к тому, что возрастет число интерполяционных узлов.

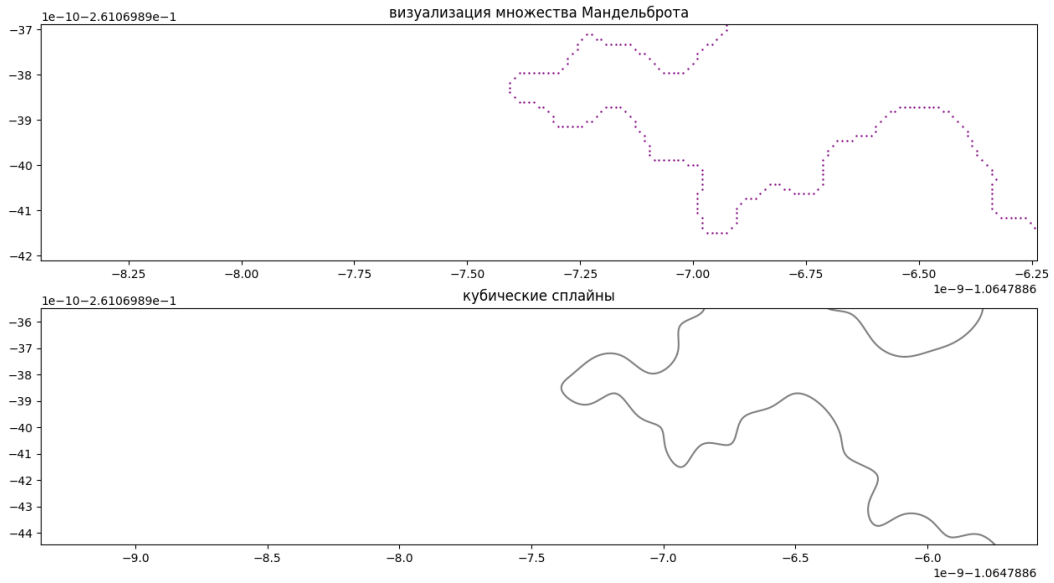

Рис. 4: приближено: множество точек контура Мандельброта (сверху) и кусочно-заданные сплайны (снизу)

Пункт 6.

функция для вычисления расстояний между точками сплайна и исходными точками множества Мандельброта

Функция, выполняющая пункт 6 (листинг 5).

Листинг 5: функция вычисления ρ [($\tilde{x}(t_i)$, $\tilde{y}(t_i)$), ($x(t_i)$, $y(t_i)$)]

```
def find_distance(point_coordinates_spline, point_coordinates_mandelbrot):
    n = len(point_coordinates_spline[0])
    distances = np.zeros(n)
    for i in range(0, n):
        x1, y1 = point_coordinates_spline[0][i], point_coordinates_spline[1][i]
        x2, y2 = point_coordinates_mandelbrot[0][i],
point_coordinates_mandelbrot[1][i]
        distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
        distances[i] = distance
    return distances

distances = find_distance(spline_points, mandelbrot_points)
print(f"среднее расстояние: {np.mean(distances)}")
print(f"стандартное отклонение: {np.mean(np.std(distances))}")
```

Фрагмент работы функции `find_distance(point_coordinates_spline, point_coordinates_mandelbrot)` :

среднее расстояние: 9.976604288833663e-12

стандартное отклонение: 6.6588964671664e-12

Пункт 7.

Функция, выполняющая всю базовую часть

Функция `lab1_base(filename_in, factor, filename_out)`, которая выполняет всю базовую часть представлена в листинге 6. Она получает на вход файл с исходным множеством точек, значение разреженности, выходной файл для записи коэффициентов сплайнов. Данная функция полностью реализует базовую часть задания: построение разреженного множества интерполяционных узлов, формирование матрицы для решения матричного уравнения (формула 3), нахождение коэффициентов сплайнов, их запись в файл, нахождение расстояний между точками исходного множества и точками, принадлежащими сплайну.

Листинг 6: функция, выполняющая базовую часть задания

```
def lab1_base(filename_in, factor, filename_out):
    plt.subplot(2, 1, 1)
    plt.axis('equal')
    mandelbrot_points = plot_mandelbrot_points(filename_in)
    sparse_matrix = create_sparse_matrix(filename_in, factor)
    x = [sparse_matrix[0], sparse_matrix[1]]
    y = [sparse_matrix[0], sparse_matrix[2]]
    A_x = get_matrix_A(x)
    A_y = get_matrix_A(y)
    b_x = get_matrix_b(x)
    b_y = get_matrix_b(y)
    c_x = gauss_method(A_x, b_x)
    c_y = gauss_method(A_y, b_y)
    xcoefs = get_coefs_spline(x, c_x)
    ycoefs = get_coefs_spline(y, c_y)
    coefs = np.concatenate((xcoefs, ycoefs), 1)
    np.savetxt(filename_out, coefs, delimiter=' ')
    plt.subplot(2, 1, 2)
    spline_points = show_cube_splines(xcoefs, ycoefs, factor)
    distances = find_distance(spline_points, mandelbrot_points)
    plt.show()
```

Продвинутая часть.

Пункт 8.

Реализация класса для автоматического дифференцирования

Класс для автоматического вычисления производной с помощью концепции дуальных чисел представлен в листинге 7. В классе перегружены математические операторы сложения, вычитания, умножения и возведения в степень с помощью магических методов, согласно арифметике дуальных чисел (таблица 1).

Дуальное[2] число $\langle a, b\rangle = a + b\epsilon$, где $a, b \in \mathbb{R}$, тогда как ϵ абстрактное число, такое, что $\epsilon \triangleq 0$, но $\epsilon \neq 0$.

$$\langle a, b\rangle + \langle c, d\rangle = a + b\epsilon + c + d\epsilon = \langle a + c, b + d\rangle$$

$$\langle a, b\rangle \times \langle c, d\rangle = (a + b\epsilon)(c + d\epsilon) = ac + (bc + ad)\epsilon + bd\epsilon^2 = \langle ac, bc + ad\rangle$$

$$\langle a, b\rangle^n = (a + b\epsilon)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}(b\epsilon)^k = a^n + na^{n-1}b\epsilon$$

Табл. 1: арифметика дуальных чисел

Листинг 7: класс для автоматического дифференцирования

```
class DualNum:
    def __init__(self, real, dual):
        self.real = real
        self.dual = dual

    def __add__(self, other):
        if isinstance(other, DualNum):
            return DualNum(self.real + other.real, self.dual + other.dual)
        else:
            return DualNum(self.real + other, self.dual)

    def __sub__(self, other):
        if isinstance(other, DualNum):
            return DualNum(self.real - other.real, self.dual - other.dual)
        else:
            return DualNum(self.real - other, self.dual)

    def __mul__(self, other):
        if isinstance(other, DualNum):
            return DualNum(self.real * other.real, self.real * other.dual +
self.dual * other.real)
        else:
            return DualNum(self.real * other, self.dual * other)

    def __pow__(self, power):
        if isinstance(power, int):
```

```
            return DualNum(self.real ** power, power * self.real ** (power - 1)
* self.dual)
        else:
            raise ValueError("степень должна быть целой")

    def __str__(self):
        return f"{self.real} + ε{self.dual}"
```

Пункт 9.

Реализация функции автоматического расчёта первой производной кубического сплайна

Функция, выполняющая пункт 9 представлена в листинге 8.

Вид кубического сплайна $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$

Функция принимает на вход коэффициенты сплайна в виде массива [n][4], точку, в которой должна продифференцироваться функция, и x_i. Функция возвращает дуальную составляющую результата, являющуюся численным значением производной сплайна в заданной точке соответствующего сплайна.

Листинг 8: функция автоматического расчёта первой производной кубического сплайна

```
def calculate_derivative(coefficients, x, x_i):
    x_dual = DualNum(x, 1)
    f = DualNum(coefficients[0], 0)
    for i in range(1, len(coefficients)):
        f += DualNum(coefficients[i], 0) * (x_dual - DualNum(x_i, 0)) ** i
    return f.dual
```

Пункт 10.

Реализация функции построения нормали к вектору

Функция, выполняющая пункт 10 представлена в листинге 9. В функцию передаются координаты 2 точек, характеризующие касательную к сплайну (вектор $G(t) = \frac{d}{dt}(\tilde{x}(t),\ \tilde{y}(t))$).

Нормаль находится по формуле (4), подставив в нее некоторый x и значение угла наклона касательной вместо 1 производной функции сплайна в точке касания. Первая точка перпендикуляра является точкой касания и геометрической серединой отрезка касательной

$$y_н = f(x_0) - \frac{1}{f'(x_0)}(x - x_0) \qquad (4)$$

Где f `(x_0) – значение 1 производной функции сплайна в точке касания,
f (x_0) – ордината сплайна в точке касания
x_0 – абсцисса точки касания

Листинг 9: функция для построения нормали к вектору

```
def find_normal(x, y):
    perpendicular_x = [0, 0]
    perpendicular_y = [0, 0]
    centre_x = x[0] + (x[1]-x[0])/2
    centre_y = y[0] + (y[1]-y[0])/2
    k = (y[1] - y[0]) / (x[1] - x[0]) #вычисление наклона касательной
    rnd_x = 0.01
    x2 = centre_x + rnd_x #случайный x
    y2 = centre_y - 1 / k * (x2-centre_x)
    maxsize = 0.0000000002 # максимальный размер перпендикуляра
    while math.sqrt((x2-centre_x)**2 + (y2-centre_y)**2) > maxsize:
        rnd_x /= 2
        x2 = centre_x + rnd_x
        y2 = centre_y - 1 / k * rnd_x
    perpendicular_x[0], perpendicular_x[1] = centre_x, x2
    perpendicular_y[0], perpendicular_y[1] = centre_y, y2
    return perpendicular_x, perpendicular_y
```

Пункт 11.

Построение векторов $G(t_j)$ и нормали к каждому из них $R(t_j)$

Функция, выполняющая пункт 11 представлена в листинге 10. Получая на вход коэффициенты сплайна и значения производных в интерполяционных узлах, строит касательные и нормали по 2 точкам, в некоторых узлах.

Листинг 10: функция для отображения нормали и касательной к точкам сплайнов

```
def plot_vector_and_normal(xcoefs, ycoefs, xderivatives, yderivatives):
    n = len(xderivatives)
    for i in range (1,n, 10):
        a = xcoefs[i][0] #извлекаем коэффициенты
        t0 = i
        t = np.linspace(t0-1, t0+1, 2)
        x = a + 1*xderivatives[i]*(t-t0)
        a = ycoefs[i][0]
        y = a + 1*yderivatives[i]*(t-t0)
        perpendicular_x, perpendicular_y = find_normal(x, y)
        plt.plot(perpendicular_x, perpendicular_y, 'green')
        plt.plot(x, y, 'red')
```

Результат работы функции отображен на рис. 5, 6.

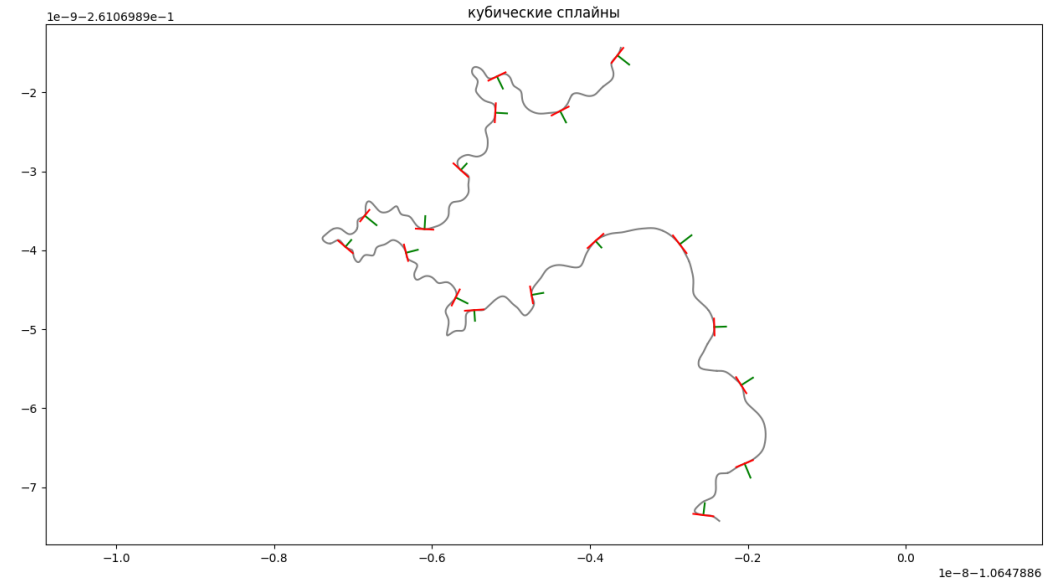

Рис. 5: результат работы функции plot_vector_and_normal ()

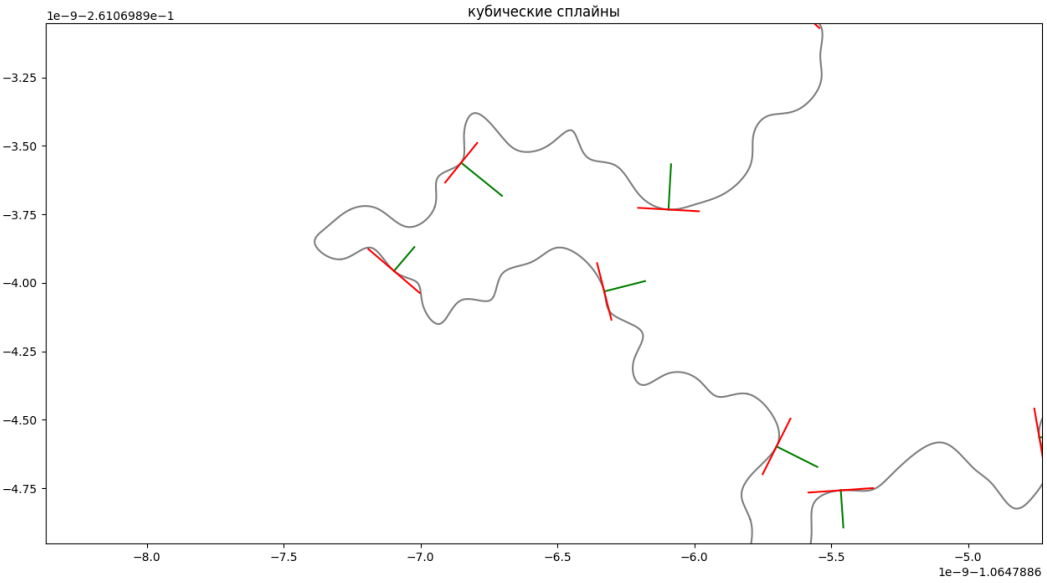

Рис. 6: результат работы функции plot_vector_and_normal (приближено)

Заключение

В результате был освоен такой метод кусочной интерполяции, как построение кубических сплайнов, а также метод автоматического дифференцирования, основанный на концепции дуальных чисел. Было проинтерполировано некоторое множество Мандельброта, получена визуально гладкая кривая, проходящая через некоторые точки этого множества, была высчитана ошибка интерполяции, которая оказалась невысокой. При помощи автоматического

дифференцирования были построены касательные и нормали к некоторым точкам сплайна.

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. Соколов, А.П. Инструкция по выполнению заданий к семинарским занятиям (общая). Москва: Соколов, А.П., 2018-2022. С. 7. URL: https://archrk6.bmstu.ru. (облачный сервис кафедры РК6).
4. Першин А.Ю. Сборник задач семинарских занятий по курсу «Вычислительная математика»: Учебное пособие. / Под редакцией Соколова А.П. [Электронный ресурс]. Москва, 2018-2021. С. 20. URL: https://archrk6.bmstu.ru. (облачный сервис кафедры РК6).
5. Першин А.Ю., Соколов А.П. Сборник постановок задач на лабораторные работы по курсу «Вычислительная математика»: Учебное пособие. [Электронный ресурс]. Москва, 2021. С. 54. URL: https://archrk6.bmstu.ru. (облачный сервис кафедры РК6)