МГТУ им. Н.Э.Баумана, кафедра «Системы автоматизированного проектирования» (РК-6), шаблон LAB (размещение sa2tml)

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:	Зотов Даниил Александрович
Группа:	РК6-53Б
Тип задания:	лабораторная работа
Тема:	Интерполяция в условиях измерений с неопределенностью

Студент ____________ Зотов Д. А.
подпись, дата Фамилия, И.О.

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

Москва, 2021

Содержание

Интерполяция в условиях измерений с неопределенностью

1 Задание

Базовая часть:

1. *Разработать функцию qubic_spline_coeff(x_nodes, y_nodes), которая посредством решения матричного уравнения вычисляет коэффициенты естественного кубического сплайна. Для простоты, решение матричного уравнения можно производить с помощью вычисления обратной матрицы с использованием функции numpy.linalg.inv().*

2. *Написать функции qubic_spline(x, qs_coeff) и d_qubic_spline(x, qs_coeff), которые вычисляют соответственно значение кубического сплайна и его производной в точке x (qs_coeff обозначает матрицу коэффициентов).*

3. *Используя данные в таблице 1, требуется построить аппроксимацию зависимости уровня поверхности жидкости h(x) от координаты x (см. рисунок 1) с помощью кубического сплайна и продемонстрировать ее на графике вместе с исходными узлами.*

Таблица 1. Значения уровня поверхности вязкой жидкости (рис. 1)

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
h_i	3.37	3.95	3.73	3.59	3.15	3.15	3.05	3.86	3.60	3.70	3.02

Продвинутая часть:

1. *Разработать функцию l_i(i, x, x_nodes), которая возвращает значение i-го базисного полинома Лагранжа, заданного на узлах с абсциссами x_nodes, в точке x.*

2. *Написать функцию L(x, x_nodes, y_nodes), которая возвращает значение интерполяционного полинома Лагранжа, заданного на узлах с абсциссами x_nodes и ординатами y_nodes, в точке x.*

3. *Известно, что при измерении координаты x_i всегда возникает погрешность, которая моделируется случайной величиной с нормальным распределением с нулевым математическим ожиданием и стандартным отклонением 10^{-2}. Требуется провести следующий анализ, позволяющий выявить влияние этой погрешности на интерполяцию:*

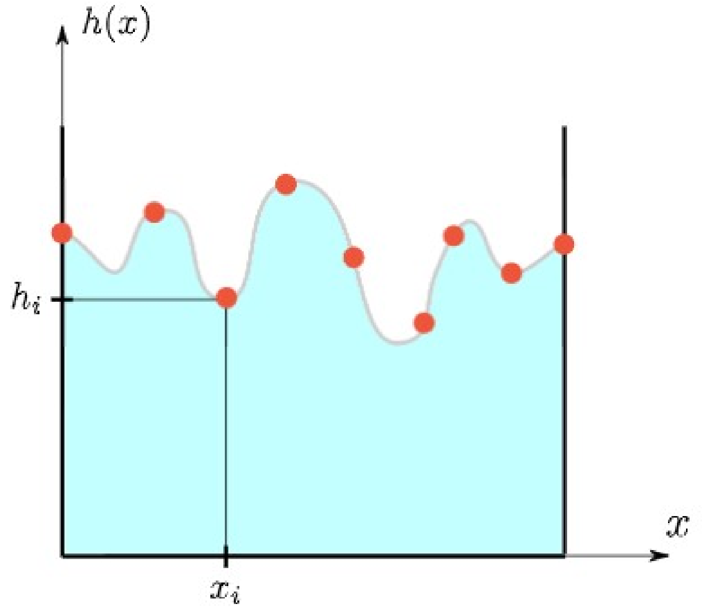


Рис. 1. Поверхность вязкой жидкости (серая кривая), движущейся сквозь некоторую среду (например, пористую). Её значения известны только в нескольких точках (красные узлы).

(a) *Сгенерировать 1000 векторов значений* $[\tilde{x}_0, ..., \tilde{x}_{10}]^T$*, предполагая, что* $\tilde{x}_i = x_i + Z$*, где* x_i *соответствует значению в таблице 1 и* Z *является случайной величиной с нормальным распределением с нулевым математическим ожиданием и стандартным отклонением* 10^{-2}.

(b) *Для каждого из полученных векторов построить интерполянт Лагранжа, предполагая, что в качестве абсцисс узлов используются значения* $\tilde{x}_i$*, а ординат –* h_i *из таблицы 1. В результате вы должны иметь 1000 различных интерполянтов.*

(c) *Предполагая, что все интерполянты представляют собой равновероятные события, построить такие функции* $\tilde{h}_l(x)$ *и* $\tilde{h}_u(x)$*, где* $\tilde{h}_l(x) < \tilde{h}_u(x)$ *для любого* $x \in [0;1]$*, что вероятность того, что значение интерполянта в точке* x *будет лежать в интервале* $[\tilde{h}_l(x); \tilde{h}_u(x)]$ *равна 0.9.*

(d) *Отобразить на едином графике функции* $\tilde{h}_l(x)$, $\tilde{h}_u(x)$*, усредненный интерполянт и узлы из таблицы 1.*

(e) *Какие участки интерполянта и почему являются наиболее чувствительными к погрешностям?*

4. *Повторить анализ, описанный в предыдущем пункте, в предположении, что координаты* x_i *вам известны точно, в то время как измерения уровня поверхности* h_i *имеют ту же погрешность, что и в предыдущем пункте. Изменились ли выводы вашего анализа?*

5. *Повторить два предыдущие пункта для случая интерполяции кубическим сплайном. Какие выводы вы можете сделать, сравнив результаты анализа для интерполяции Лагранжа и интерполяции кубическим сплайном?*

6. *Опциональное задание. Изложенный выше анализ позволяет строить доверительные интервалы исключительно для интерполянтов, не оценивая доверительные интервалы с точки зрения предсказаний значений между узлами. Интересным методом интерполяции, позволяющим получить именно такие вероятностные оценки, является регрессия на основе гауссовских процессов, известная также как кригинг. В этом опциональном задании предлагается провести интерполяцию по данным из таблицы 1, используя кригинг.*

2 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, а также интерполяции полиномами Лагранжа, сравнение полученных результатов.

3 Выполненные задачи

1. Разработка функции вычисления коэффициентов естественного кубического сплайна;

2. Разработка функции для вычисления значения кубического сплайна и его производной в точке x, а затем, построение графика интерполяции кубическими сплайнами;

3. Разработка функции вычисления базисного полинома Лагранжа в точке x и интерполяционного полинома Лагранжа в точке x;

4. Генерация 1000 векторов с добавлением к исходным узлам случайной величины для абсцисс, а затем для ординат и построение 1000 графиков с помощью глобальной интерполяции Лагранжа;

5. Построение доверительного интервала и усредненного интерполянта для интерполянтов Лагранжа и анализ проведенных построений;

6. Проведение повторного анализа, проведенного с интерполянтами Лагранжа для интерполяции кубическими сплайнами и сравнение результатов.

4 Интерполяция кубическими сплайнами

1.1. Разработка функции для вычисления коэффициентов естественного кубического сплайна

Интерполяция кубическими сплайнами является одним из самых популярных видов локальной интерполяции. 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:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \tag{1}$$

где a_i, b_i, c_i, d_i - коэффициенты кубического сплайна.

$S(x) = \{ S_i(x) : x \in [x_i; x_{i+1}] \}$

Матричное уравнение для коэффициентов имеет вид: [?] ссылка на источник данных...

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

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

где $h_i = x_{i+1} - x_i$.

Для разработки функции вычисления коэффициентов кубического сплайна *qubic_spline_coeff(x_nodes, y_nodes)* необходимо на основе интерполяционных узлов *x_nodes, y_nodes* заполнить элементы матричного уравнения (2).

Алгоритм ее заполнения следующий:

1. Создать нулевую матрицу размером n x n, где n - количество интерполяционных узлов. Данная матрица будет заполнена в соответствии с первой матрицей из (2). Для создания нулевой матрицы используем функцию *numpy.zeros*, имеющую вид:

```
numpy.zeros(shape, dtype=float, order='C', *, like=None)
```

```
shape: int или tuple of ints - размерность нулевой матрицы;
```

```
dtype: data-type - желаемый тип выходной матрицы;
order: {'C', 'F'} - параметр, показывающий, как хранить многомерные
данные ('C' - в порядке строк, 'F' - в порядке столбцов;
like: array_like - ссылочный объект, позволяющий создавать массивы,
не являющиеся массивами numpy.
```

2. Создать нулевой вектор размером n. Данный вектор будет результирующим вектором из (2). Для создания нулевого вектора используем функцию numpy.zeros;

3. Заполнить элементы нулевой матрицы $[1,1]$ и $[n,n]$ единицами в соответствии с (2);

4. В цикле продолжить заполнение элементов строк $i = 2, ..., n-1$ в соответствии со следующим алгоритмом: со смещением заполняются элементы по бокам от главной диагонали в соответствии с (2), затем заполняется элемент главной диагонали, являющийся удвоенной суммой боковых элементов, каждую итерацию смещение трех элементов увеличивается на 1. Боковые элементы имеют вид $h1 = x\_nodes[i] - x\_nodes[i-1]$ и $h2 = x\_nodes[i+1] - x\_nodes[i]$ соответствнно;

5. В цикле заполнить элемент результирующего вектора, соответствующий данной итерации в соответствии с формулами из (2). Элемент на каждой итерации имеет вид $(3/h1) * (y\_nodes[i+1] - y\_nodes[i]) - (3/h2) * (y\_nodes[i] - y\_nodes[i-1])$.

В итоге, получим заполненную матрицу и результирующий вектор. Решим матричное уравнение типа $AX = B$, где A - заполненная матрица, X - вектор-столбец коэффициентов, B - результирующий вектор. Решением для X является:

$$X = A^{-1}B. \tag{3}$$

Рассчитаем обратную матрицу A^{-1}. Для этого используем функцию *numpy.linalg.inv()*, имеющую вид:

```
numpy.linalg.inv(a)

    a: array_like - матрица, от которой необходимо получить обратную матрицу.
```

Для решения матричного уравнения (3) используем функцию *numpy.dot()*, имеющую вид:

```
numpy.dot(a, b, out=None)

    a: array-like - первый аргумент (матрица);
    b: array-like - второй аргумент (матрица);
    out: ndarray - выходной аргумент.
```

В результате решения матричного уравнения получим матрицу коэффициентов, которую будет возвращать функция.

Листинг функции *qubic_ spline_ coeff*:

```
def qubic_spline_coeff(x_nodes, y_nodes):
    main_matrix = np.zeros((int(len(x_nodes)), int(len(x_nodes))))
    result_matrix = np.zeros(len(x_nodes))
    result_matrix = result_matrix.T
    main_matrix[0, 0] = main_matrix[len(x_nodes) - 1, len(x_nodes) - 1] = 1
    counter = 0

    for i in range(1, len(x_nodes) - 1):
        main_matrix[i, counter] = h_n2 = x_nodes[i] - x_nodes[i-1]
        main_matrix[i, counter + 2] = h_n1 = x_nodes[i+1] - x_nodes[i]
        main_matrix[i, counter + 1] = 2 * (main_matrix[i, counter] + \
                    main_matrix[i, counter + 2])
        result_matrix[i] = (3 / h_n1) * (y_nodes[i+1] - y_nodes[i]) - \
                    (3 / h_n2) * (y_nodes[i] - y_nodes[i-1])
        counter += 1

    main_matrix_inv = np.linalg.inv(main_matrix)
    coeff_matrix = np.dot(main_matrix_inv, result_matrix)

    return coeff_matrix
```

1.2. Разработка функции для вычисления значения кубического сплайна и его производной в точке x

Для вычисления значения кубического сплайна в точке x по формуле (1) изначально требуется определить, к какому отрезку $[i;i+1], i = 0, ..., n-1$ принадлежит x. Это необходимо для подстановки в формулу (1) коэффициентов кубического сплайна. Чтобы определить принадлежность точки к отрезку, в цикле сделаем следующее сравнение: если x принадлежит отрезку $[x_i; x_{i+1}], i = 0, ..., n-1$, то запоминаем переменную i, если нет - то переходим к рассмотрению следующего отрезка. Также учтем, что для выполнения продвинутой части стоит рассмотреть случай экстраполяции, поэтому, если x выходит за крайний левый интерполяционный узел, то считаем, что $i = 0$ (первый отрезок интерполирования), если x выходит за крайний правый интерполяционный узел, то считаем, что $i = n-1$ (последний отрезок интерполирования). Это необходимо для того, чтобы продолжить функцию $S(x)$ вне отрезка $[a;b]$.

После получения отрезка для интерполирования, необходимо рассчитать коэффициенты для уравнения (1). Коэффициент c_i получим из вектора коэффициентов, рассчитанного в п. 1.1. Коэффициенты a_i, b_i, d_i имеют следующий вид:

$$a_i = f(x_i) \tag{3}$$

$$b_i = \frac{1}{h_i}(a_{i+1} - a_i) - \frac{h_i}{3}(c_{i+1} + 2c_i) \tag{4}$$

$$d_i = \frac{c_{i+1} - c_i}{3h_i} \tag{5}$$

Функция вычисления значения кубического сплайна в точке x будет возвращать значение $S_i(x)$ с учетом формул (1), (3), (4), (5) и значения c_i из вектора коэффициентов qs_coeff.

Производная кубического сплайна в точке x имеет вид:

$$S_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2 \tag{6}$$

Функция вычисления значения производной кубического сплайна в точке x будет возвращать значение S_i' с учетом формул (3), (4), (5), (6) и значения c_i из вектора коэффициентов qs_coeff.

Программная реализация функции представлена ниже.

Листинг функции *qubic_ spline*:

```
def qubic_spline(x, qs_coeff, x_nodes, y_nodes):
    i = -1
    for n in range(0, len(x_nodes) - 1):
        if (x >= x_nodes[n]) and (x <= x_nodes[n+1]):
            i = n
            break
    if x < x_nodes[0]:
        i = 0
    if x > x_nodes[len(x_nodes) - 1]:
        i = len(x_nodes) - 2
    a_i = y_nodes[i]
    b_i = (1 / (x_nodes[i+1] - x_nodes[i])) * (y_nodes[i+1] - \
        y_nodes[i]) - ((x_nodes[i+1] - x_nodes[i]) / 3) * \
        (qs_coeff[i+1] + 2 * qs_coeff[i])
    c_i = qs_coeff[i]
    d_i = (qs_coeff[i+1] - qs_coeff[i]) / (3 * (x_nodes[i+1] - \
                          x_nodes[i]))
    S_x = a_i + b_i * (x - x_nodes[i]) + c_i * \
       ((x - x_nodes[i]) ** 2) + d_i * ((x - x_nodes[i]) ** 3)

    return S_x
```

Листинг функции *d_ qubic_ spline*:

```
def d_qubic_spline(x, qs_coeff, x_nodes, y_nodes):
    for n in range(0, len(x_nodes)):
        if x >= x_nodes[n] and x <= x_nodes[n+1]:
            i = n
            break

    b_i = (1 / (x_nodes[i+1] - x_nodes[i])) * (y_nodes[i+1] - y_nodes[i]) - ((
    x_nodes[i+1] - x_nodes[i]) / 3) * (qs_coeff[i+1] + 2 * qs_coeff[i])
```

```
    c_i = qs_coeff[i]
    d_i = (qs_coeff[i+1] - qs_coeff[i]) / (3 * (x_nodes[i+1] - x_nodes[i]))
    S_dx = b_i + 2 * c_i * (x - x_nodes[i]) + 3 * d_i * ((x - x_nodes[i]) ** 2)

    return S_dx
}
```

1.3. Построение апроксимации зависимости уровня поверхности жидкости h(x) от координаты x с помощью кубического сплайна

Выполним построение апроксимации зависимости уровня поверхности жидкости h(x) от координаты x с помощью кубического сплайна. Для этого сначала рассчитаем вектор коэффициентов, вызвав функцию $qubic_{s}pline_{c}oeff(x\_nodes, y\_nodes)$, передав в качестве параметров x_nodes, y_nodes значения из таблицы 1.
Создадим два пустых массива x_q, y_q для хранения координат x, y типа ndarray с помощью функции *numpy.array*. Функция *numpy.array* имеет вид:

```
numpy.array(object, dtype=None, *, copy=True, order='K',
            subok=False, ndmin=0, like=None)

    Основные параметры:
    object: array_like - массив или любой элемент, представляющий интерфейс массива,
                         используется в качестве основы для создания ndarray;
    dtype: data-type - желаемый тип для хранения;
    copy: bool - булевое значение, определяющее, делать копию или создавать новый
                 объект в памяти
    like: array_like - ссылочный объект, позволяющий создавать массивы,
   не являющиеся массивами numpy.
```

С помощью функции *numpy.arange* с параметрами $start = x\_nodes[0], stop = x\_nodes[len(x\_nodes) - 1], step = 0.0005$ получим 2000 точек на оси абсцисс для построения функции и присвоим их переменной *interval* (прим. $interval = x\_q$). Функция *numpy.arange* имеет вид:

```
numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)

    start: integer или real - начало интервала, по умолчанию start=0;
    stop: integer или real - конец интервала, не включает в себя эту точку;
    step: integer или real - размер шага между значениями;
    dtype: data-type - желаемый тип для хранения;
    like: array_like - ссылочный объект, позволяющий создавать массивы,
   не являющиеся массивами numpy.
```

Для построения графика в цикле по переменной $interval$ будем вычислять значение кубического сплайна в данной точке. Для этого вызовем функцию $qubic_spline(x, qs\_coeff, x\_nodes, y\_n$ передав ей в качестве аргументов текущее значение x в цикле, и значения x_nodes, y_nodes из таблицы 1.

```
for node in interval:
    new_node = qubic_spline(node, coeff, x_nodes, y_nodes)
    y_q = np.append(y_q, new_node)
```

После расчета значений координат точек, построим апроксимацию с помощью кубического сплайна. Построение графиков произведем с помощью функций библиотеки *matplotlib*: *matplotlib.pyplot.title*, *matplotlib.pyplot.grid*, *matplotlib.pyplot.plot*, *matplotlib.pyplot.scatter*.

Функция *matplotlib.pyplot.title* имеет вид:

```
matplotlib.pyplot.title(label, fontdict=None, loc='center', pad=None, **kwargs) -
                        показать заголовок графика.

    Основные параметры:
    label: str - заголовок графика.
```

Функция *matplotlib.pyplot.grid* имеет вид:

```
matplotlib.pyplot.grid(b=None, which='major', axis='both', **kwargs) -
                       нанесение сетки на график.

    Основные параметры:
    b: bool - параметр, определяющий, показывать ли сетку;
    **kwargs : Line2D - список параметров, определяющий свойства сетки:
        alpha: scalar - значение от [0;1], определяющее прозрачность.
```

Функция *matplotlib.pyplot.plot* имеет вид:

```
matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs) -
                       осуществить построение графика.

    Основные параметры:
    x,y : array-like или scalar - значения координат, если array-like, то
                                  размерности x и y должны совпадать;
    fmt : str - строка формата (например, 'o' для точек);
    **kwargs : Line2D - список параметров, определяющий свойства графика:
        color: str - цвет графика.
```

Функция *matplotlib.pyplot.scatter* имеет вид:

```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None,
norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, *,
edgecolors=None, plotnonfinite=False, data=None, **kwargs) -
                нанесение маркеров-точек на график.

    Основные параметры:
    x,y : array-like или float - значения координат, если array-like, то
                              размерности x и y должны совпадать;
    **kwargs : Line2D - список параметров, определяющий свойства графика:
        color: str - цвет маркеров.
```

Полный листинг представлен ниже.

Листинг вызова построения графика:

```python
if __name__ == "__main__":
    x_nodes = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
    y_nodes = [3.37, 3.95, 3.73, 3.59, 3.15, 3.15, 3.05, 3.86, 3.60, 3.70, 3.02]

    coeff = qubic_spline_coeff(x_nodes, y_nodes)

    x_q = np.array([])
    y_q = np.array([])
    interval = np.arange(x_nodes[0], x_nodes[len(x_nodes) - 1], 0.0005)
    print(interval)
    x_q = interval
    for node in interval:
        new_node = qubic_spline(node, coeff, x_nodes, y_nodes)
        y_q = np.append(y_q, new_node)

    plt.title('Апроксимация зависимости уровня поверхности жидкости h(x)')
    plt.grid(alpha=0.3)
    plt.plot(x_q, y_q, color='green')
    plt.scatter(x_nodes, y_nodes, color='red')
    plt.show()
```

Полученный график представлен на рис. 2. В результате построения кубического сплайна можно заметить, что график "гладкий"и не имеет осцилляций, а также проходит через заданные интерполяционные узлы, из чего можно сделать вывод, что интерполяция кубическими сплайнами построена корректно и может быть использована для проведения дальнейшего анализа.

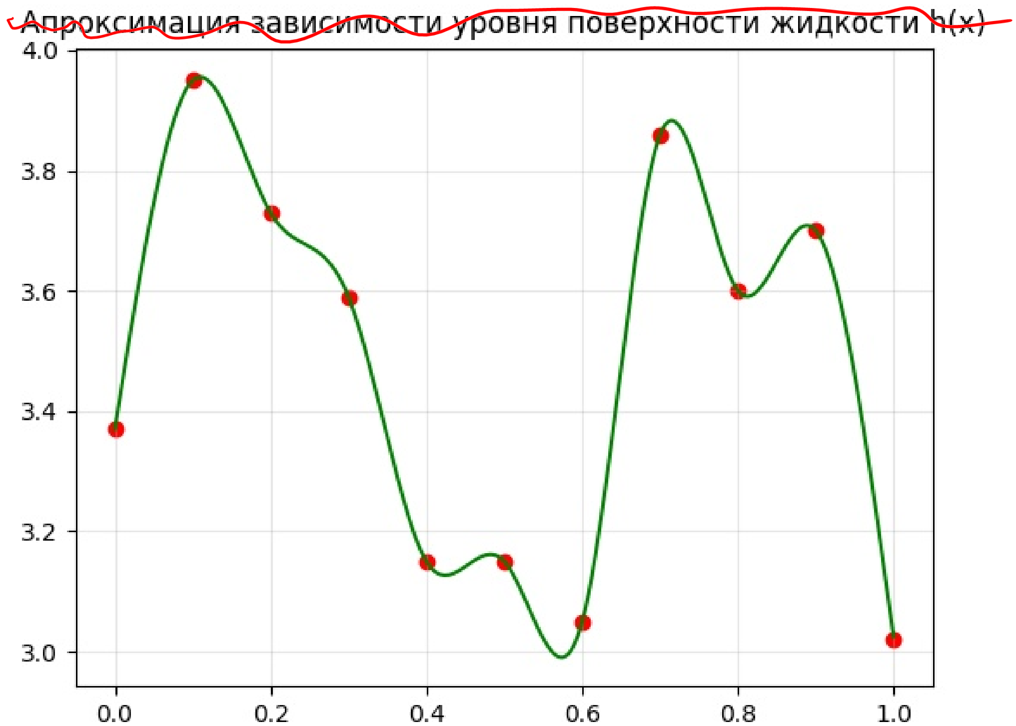


Рис. 2. Построение апроксимации зависимости уровня поверхности жидкости h(x) через 11 интерполяционных узлов с помощью кубических сплайнов.

5 Анализ влияния погрешности на интерполяцию Лагранжа и интерполяцию кубическими сплайнами

2.1. Разработка функции для базисного полинома Лагранжа

Интерполяция Лагранжа является примером глобальной интерполяции, когда интерполирующая функция $\widetilde{f}(x)$ задана на всем промежутке $[a;b]$. По определению, если функция $f(x)$ задана в n интерполяционных узлах $a = x_1, x_2, ..., x_n = b$ на отрезке $[a;b]$, то интерполяционным многочленом для функции $f(x)$ и соответствующих узлов интерполяции называется функция

$$L_{n-1}(x) = \sum_{i=1}^{n} f(x_i) l_i(x), \quad l_i(x) = \prod_{i \neq j} \frac{x - x_j}{x_i - x_j}, \tag{7}$$

где $l_i(x)$ является базисным многочленом $(n-1)$-й степени.

Для разработки функции получения базисного полинома Лагранжа в точке x воспроизведем формулу: определим переменную $l\_x = 1$ для накопления произведения, а затем, в цикле по всем значениям $j, j = 0, ..., n$ накапливаем в переменную l_x произведение $\prod_{i \neq j} \frac{x - x_j}{x_i - x_j}$, если $i \neq j$. В случае $i = j$, вызываем *continue* для перехода к следующей итерации.

Результатом работы функции будет значение l_x базисного полинома Лагранжа в точке x.

Листинг функции *l_i*:

```python
def l_i(i, x, x_nodes):
    l_x = 1

    for j in range(0, len(x_nodes)):
        if i == j:
            continue
        else:
            l_x = l_x * ((x - x_nodes[j]) / (x_nodes[i] - x_nodes[j]))

    return l_x
```

2.2. Разработка функции для интерполяционного полинома Лагранжа

Для получения значения интерполяционного полинома Лагранжа по формуле (7) необходимо вычислить сумму всех базисных полиномов Лагранжа, умноженных на значения функции $f_i(x)$для соответвующих точек.

Для разработки функции получения значения интерполяционного полинома Лагранжа воспроизведем формулу: определим переменную $L\_x = 0$ для накопления суммы, а затем, в цикле по всем значениям $i, i = 0, ..., n$ накапливаем в переменную L_x сумму из формулы (7), вызывая для получения значения базисного полинома полученную ранее функцию.

Результатом работы функции будет значение L_x интерполяционного полинома Лагранжа в точке x.

Листинг функции *L*:

```python
def L(x, x_nodes, y_nodes):
    L_x = 0

    for i in range(0, len(x_nodes)):
        f_x_i = y_nodes[i]
        L_x = L_x + f_x_i * l_i(i, x, x_nodes)

    return L_x
```

2.3а. Генерация векторов со случайной величиной

Для генерации $n = 1000$ векторов со случайной величиной Z, которая имеет нормальное распределение с нулевым математическим ожиданием и стандартным отклонением 10^{-2}, разработаем функцию $generate\_vectors(vector, scale, n)$, которая будет принимать на вход в качестве параметров исходный вектор, стандартное отклонение и количество векторов.

Для реализации функции, в цикле от 0 до n будем добавлять во внутреннем цикле к каждому значению в векторе случайную величину Z. Случайная величина Z генерируется с помощью функции $numpy.random.normal$. Функция $numpy.random.normal$ имеет вид:

```
random.normal(loc=0.0, scale=1.0, size=None)

    loc: float или array_like of floats - математическое ожидание,
                                           в нашем случае = 0;
    scale: float или array_like of floats - стандартное отклонение,
                                           в нашем случае=scale;
    size: int или tuple of ints - выходная размерность.
```

Листинг функции *generate_vectors*:

```
def generate_vectors(vector, scale, n):
    res = []

    for i in range(0, n):

        res_iteration = []
        for j in range(0, len(vector)):
            Z = np.random.normal(0, scale)
            new_elem = vector[j] + Z
            res_iteration.append(new_elem)
        res.append(res_iteration)
    result = np.array(res)

    return result
```

Для выполнения задания, необходимо вызвать функцию $generate\_vectors(x\_nodes, 10**(-2), 1000)$ для генерации 1000 векторов со стандартным отклонением 10^{-2} и математическим ожиданием 0 с использованием значений x_nodes из таблицы 1. Присвоим результат выполнения функции переменной x_tilde

2.3б. Построение интерполянта Лагранжа на абсциссах с добавлением случайной величины

Для построения интерполянтов Лагранжа разработаем функцию $make\_interpolants\_x(x\_nodes\_res, x\_nodes, y\_nodes)$, которая будет принимать на вход в качестве параметров список из векторов, изначальный вектор абсцисс и изначальный вектор ординат с использованием значений из таблицы 1.

Алгоритм работы построения графиков выглядит следующим образом:

1. Определить цикл для всех векторов X_i, $i = 0, ..., len(x\_nodes\_res) - 1$ (программно до $len(x\_nodes\_res)$):

   ```
   for n in range(len(x_nodes_res))
   ```

2. Создать пустой список $res\_iterate$ для хранения значений по ординатам;

3. Задать интервал значений по абсциссам в переменную $interval$ с помощью $numpy.arange$. Так как необходим диапазон $[0; 1]$, а параметр $stop$ не включает в себя это значение, зададим функцию как $numpy.arange(0, 1.0001, 0.005)$, задав второй параметр до 4 знака после запятой. В результате получим 200 точек для построения;

4. Преобразуем $interval$ к списку и занесем результат в переменную $interval\_list$. Для этого воспользуемся стандартной функцией $toList()$.

5. Во внутреннем цикле для каждого значения абсциссы вычислим значение ординаты, как:

   ```
       for m in interval:
           value = L(m, x_nodes_res[n], y_nodes)
           res_iterate.append(value)
   ```

6. Построим график с помощью $matplotlib.pyplot.plot$, передав функции значения $interval\_list$ (значения абсцисс), $res\_iterate$ (значения ординат);

7. Продолжить построение в цикле для всех оставшихся векторов;

8. С помощью $matplotlib.pyplot.scatter$ нанести точки-маркеры исходных значений из таблицы 1;

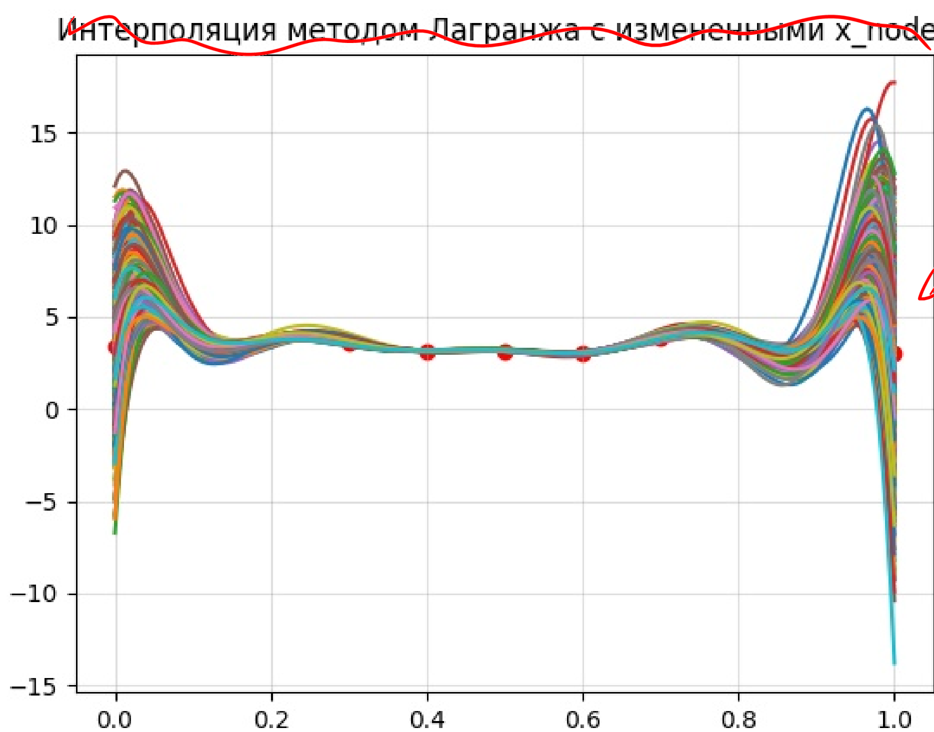
9. С помощью $matplotlib.pyplot.title, matplotlib.pyplot.grid$ нанести на график заголовок и сетку.

В результате работы функции получим необходимое количество графиков. Алгоритм построения графиков будет использоваться в дальнейшем с незначительными изменениями.

Листинг функции *make_interpolants_x*:

```python
def make_interpolants_x(x_nodes_res, x_nodes, y_nodes):
    for n in range(len(x_nodes_res)):
        res_iterate = []

        interval = np.arange(0.00, 1.0001, 0.005)
        interval_list = interval.tolist()
        for m in interval:
            value = L(m, x_nodes_res[n], y_nodes)
            res_iterate.append(value)

        plt.plot(interval_list, res_iterate)

    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Интерполяция методом Лагранжа с измененными x_nodes')
    plt.grid(alpha=0.4)
    plt.show()
```

Вызовем функцию $make\_interpolants\_x(x\_tilde, x\_nodes, y\_nodes)$. Получим 1000 графиков для интерполянта Лагранжа с использованием векторов из п. 2.3a:



плохое разрешение

Рис. 3. Интерполяция методом Лагранжа с использованием векторов из п. 2.3a.

?

2.3в, г. Построение функций $\widetilde{h}_l(x), \widetilde{h}_u(x)$ и медианного значения

По заданию необходимо построить такие функции $\widetilde{h}_l(x), \widetilde{h}_u(x)$, что $\widetilde{h}_l(x) < \widetilde{h}_u(x)$ и вероятность того, что значение интерполянта в точке $x, x \in [0;1]$ будет лежать в интервале $[\widetilde{h}_l(x); \widetilde{h}_u(x)]$ равна 0.9. Учитывая, что все интерполянты являются равновероятными событиями и интерполянты были построены согласно нормалному закону распределения, подобный доверительный интервал можно построить следующим образом: двигаясь в цикле по значениям абсцисс, используя сортировку по возрастанию, отсортировать значения ординат для каждой точки, затем взять такие значения ординат, что между ними находится $n_t = 0.9 * n = 900$ точек. Программно, это значения ординат с индексами [49], [948].

Разработаем функцию $make\_trust\_interval\_x(x\_nodes, y\_nodes, x\_nodes\_res)$. Она также будет строить усредненный интервал. Медианное значение рассчитывается как частное суммы всех значений на их количество. Будем использовать следующий алгоритм:

1. Задать интервал значений по абсциссам в переменную $interval\_x$ с помощью $numpy.arange$. Зададим функцию как $numpy.arange(0, 1.0001, 0.005)$, задав второй параметр до 4 знака после запятой. В результате получим 200 точек для построения;

2. Задать три пустых списка: $h\_l, h\_u, median$. В них будем заносить ординаты точек;

3. В цикле по значениям из $interval\_x$ задать пустой список $temp$, который с каждой новой итерации будет создаваться заново и заполняться значениями ординат для всех 1000 интерполянтов. Этот список необходим для сортировки по возрастанию;

4. Во внутреннем цикле рассчитаем значения ординат для текущей точки и занесем их в список $temp$:

```
for i in range(len(x_nodes_res)):
    f = L(x, x_nodes_res[i], y_nodes)
    temp.append(f)
```

5. Отсортировать список $temp$ стандартной функцией $sorted$ по возрастанию и добавить в списки h_l, h_u значения по индексу [49], [948] из списка $temp$. Также добавить к списку $median$ значение ординаты медианы в этой точки. Значение ординаты для медианного значения можно рассчитать с помощью функции $statistics.median$. Функция имеет вид:

   ```
   statistics.median(data)

       data: array или array-like - входной набор данных.
   ```

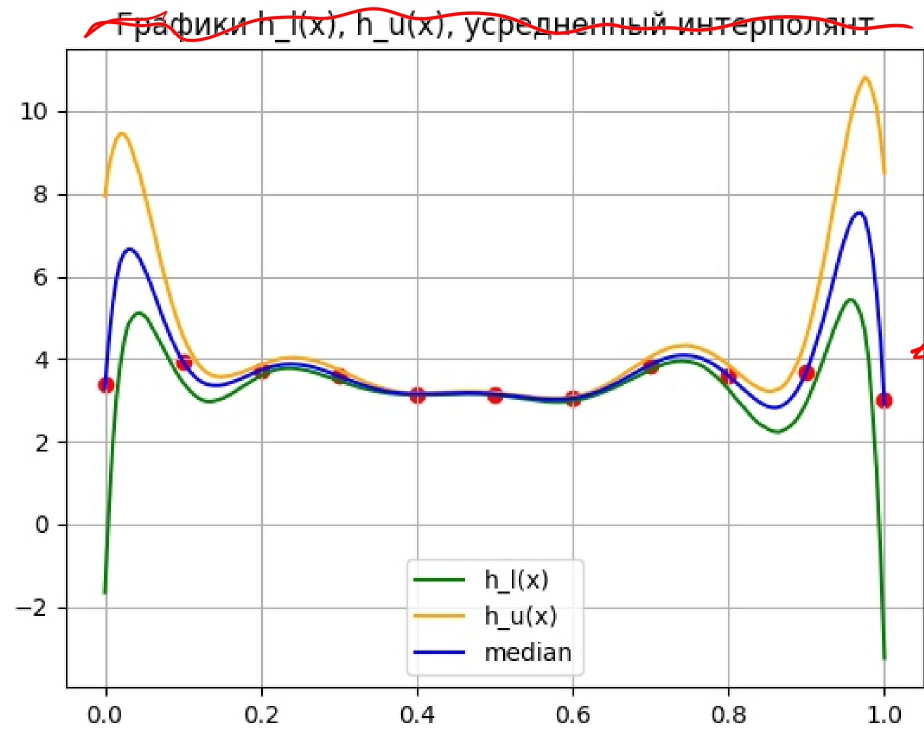
6. После завершения цикла построить графики. В качестве абсцисс используются значения $interval\_x$, в качестве ординат - значения из списков $h\_l, h\_u, median$.

Листинг функции *make_ trust_ interval_ x*:

```
def make_trust_interval_x(x_nodes, y_nodes, x_nodes_res):
    interval_x = np.arange(0.00, 1.0001, 0.005)

    h_l = []
    h_u = []
    median = ы[]

    for x in interval_x:
        temp = []
        for i in range(len(x_nodes_res)):
            f = L(x, x_nodes_res[i], y_nodes)
            temp.append(f)
        temp = sorted(temp)
        h_l.append(temp[49])
        h_u.append(temp[948])
        median.append(statistics.median(temp))

    plt.plot(interval_x, h_l, color='green', label='h_l(x)')
    plt.plot(interval_x, h_u, color='orange', label='h_u(x)')
    plt.plot(interval_x, median, color='blue', label='median')
    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Графики h_l(x), h_u(x), усредненный интерполянт')
    plt.grid()
    plt.legend()
    plt.show()
```

пробел не нужен

Вызвав функцию $make\_trust\_interval\_x(x\_nodes, y\_nodes, x\_tilde)$ получим три графика: $\widetilde{h}_l(x), \widetilde{h}_u(x)$ и усредненный интерполянт:



разрешение
↑ 200 dpi

Рис. 4. Графики $\widetilde{h}_l(x), \widetilde{h}_u(x)$ и усредненный интерполянт с узлами x_tilde

?

2.3д. Наиболее чувствительные к погрешности участки интерполянта

Как можно заметить из предыдущих построений, чувствительность интерполянтов к погрешности возрастает с приближением к концам отрезка, где заметно сильное отклонение от нуля. Следовательно, можно утверждать, что наиболее чувствительными к погрешности участками интерполянта являются участки у концов отрезка $[0; 1]$.

Данное явление называется феноменом Рунге. Феномен Рунге проявляется в возникновении осцилляций при интерполяции полиномами высоких степеней, то есть, при увеличении количества интерполяционных узлов, осцилляций у концов отрезка становится только больше, поэтому увеличение количества узлов не уменьшит осцилляций. Погрешность такой интерполяции можно уменьшить, используя интерполяционные узлы Чебышева, так как погрешность интерполяции по ним равномерно убывает для любой непрерывной функции.

2.4. Проведение анализа 2.3 с измененными значениями ординат

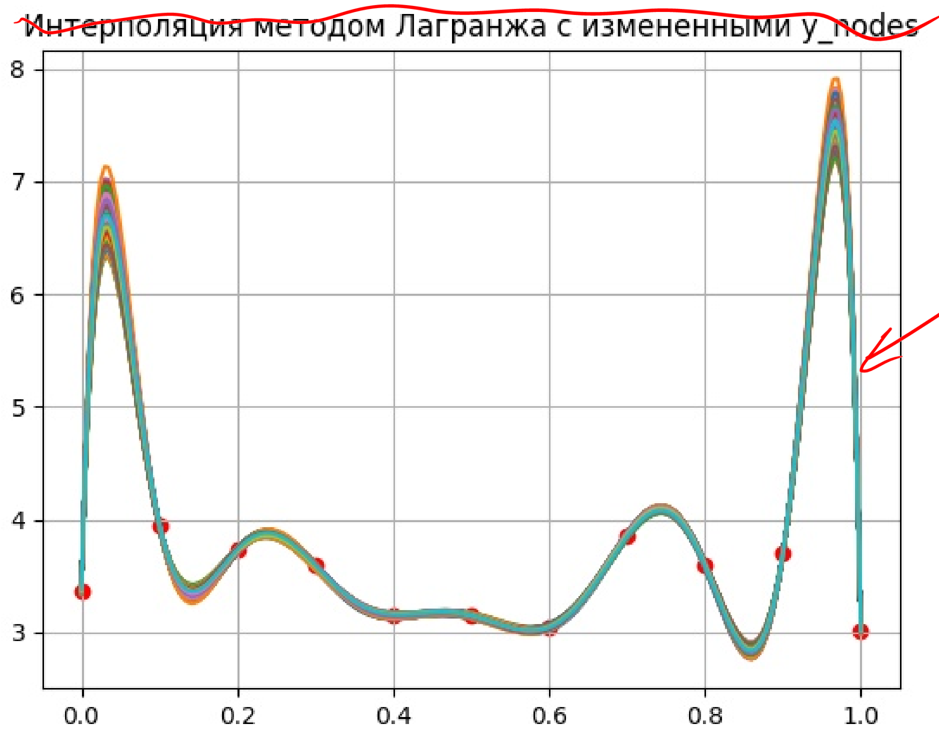
Аналогично произведем генерацию векторов со случайной величиной Z для значений ординат из таблицы 1, вызвав функцию $generate\_vectors(y\_nodes, 10 * *(-2), 1000)$. Присвоим результат выполнения функции переменной y_tilde.

Построим 1000 интерполянтов Лагранжа на промежутке $x \in [0; 1]$ со значениями y_tilde. Для построения будем использовать алгоритм, подобный алгоритму, описанному в п. 2.3б. Единственное отличие алгоритма будет в том, что во внутрненнем цикле функции L передается список x_nodes из таблицы 1, а список ординат - x_tilde. Разработаем функцию $make\_interpolants\_y(y\_nodes\_res, x\_nodes, y\_nodes)$.

Листинг функции *make_interpolants_y*:

```
def make_interpolants_y(y_nodes_res, x_nodes, y_nodes):
    for n in range(len(y_nodes_res)):
        res_iterate = []
        interval = np.arange(0.00, 1.0001, 0.005)
        interval_list = interval.tolist()

        for m in interval:
            value = L(m, x_nodes, y_nodes_res[n])
            res_iterate.append(value)

        plt.plot(interval_list, res_iterate)

    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Интерполяция методом Лагранжа с измененными y_nodes')
    plt.grid()
    plt.show()
```

Результат вызова функции $make\_interpolants\_y(y\_tilde, x\_nodes, y\_nodes)$:



Разрешение ↑ 200 dpi

Рис. 5. Интерполяция методом Лагранжа с использованием векторов с измененными y_nodes

Построим доверительный интервал и усредненный интерполянт, задав функции $\widetilde{h}_l(x), \widetilde{h}_u(x)$, что $\widetilde{h}_l(x) < \widetilde{h}_u(x)$ по алгоритму, описанному в п. 2.3в, г. Отличие алгоритма будет в том, что функции L передается список x_nodes из таблицы 1, а список ординат - x_tilde. Разработаем функцию $make\_trust\_interval\_y(x\_nodes, y\_nodes, y\_nodes\_res)$.

Листинг функции *make_trust_interval_y*:

```
def make_trust_interval_y(x_nodes, y_nodes, y_nodes_res):
    interval_x = np.arange(0.00, 1.0001, 0.005)

    h_l = []
    h_u = []
    median = []

    for x in interval_x:
        temp = []
        for i in range(len(y_nodes_res)):
            f = L(x, x_nodes, y_nodes_res[i])
            temp.append(f)
```

```
        temp = sorted(temp)
        h_l.append(temp[48])
        h_u.append(temp[947])
        median.append(statistics.median(temp))

    plt.plot(interval_x, h_l, color='green', label='h_l(x)')
    plt.plot(interval_x, h_u, color='orange', label='h_u(x)')
    plt.plot(interval_x, median, color='blue', label='median', linewidth='1')
    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Графики h_l(x), h_u(x), усредненный интерполянт')
    plt.grid()
    plt.legend()
    plt.show()
```

Результат вызова функции $make\_trust\_interval\_y(x\_nodes, y\_nodes, y\_nodes\_res)$:

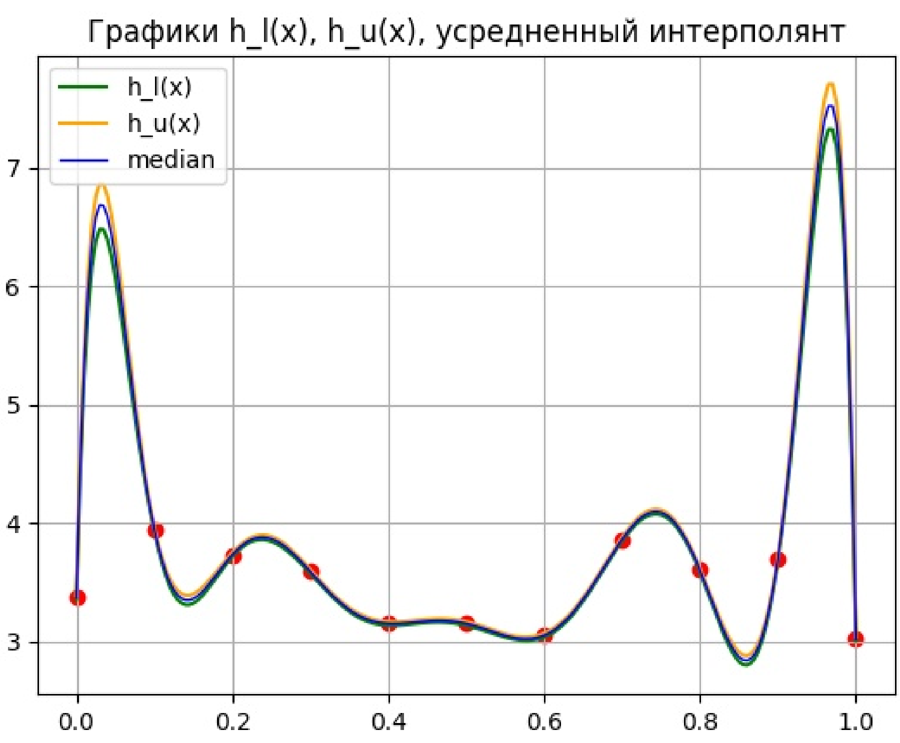


Рис. 6. Графики $\widetilde{h}_l(x)$,$\widetilde{h}_u(x)$ и усредненный интерполянт с узлами y_tilde

Разница между проведением анализа 2.3 и 2.4 состоит в том, что доверительный интервал на рис. 6. заметно сузился на $[0;1]$ по сравнению с рис. 4. Можно утверждать, что ординаты с погрешностью вносят меньший вклад в ошибку интерполяции, чем погрешность значения абсцисс, и результат интерполяции более приемлем засчет того, что при вычислении базисного полинома Лагранжа значение ординаты всегда имеет первую степень, в отличие от значений абсцисс. Однако, эффект Рунге все равно наблюдается.

2.5. Проведение анализа 2.3, 2.4 для интерполяции кубическими сплайнами

Повторим эксперименты 2.3, 2.4 для интерполяции кубическими сплайнами. Построение кубических сплайнов с узлами x_tilde, будет производиться по аналогичному алгоритму из 2.3б, за исключением того, что теперь для расчета ординат вызывается функция $qubic\_spline$. Разработаем функцию $make\_qubic\_splines\_x(x\_nodes, y\_nodes, x\_nodes\_res)$.

Листинг функции *make_qubic_splines_x*:

```python
def make_qubic_splines_x(x_nodes, y_nodes, x_nodes_res):
    for n in range(len(x_nodes_res)):
        c = qubic_spline_coeff(x_nodes_res[n], y_nodes)
        interval_x = np.arange(0.00, 1.0001, 0.005)
        res_iterate = []

        for x in interval_x:
            S = qubic_spline(x, c, x_nodes_res[n], y_nodes)
            res_iterate.append(S)

        interval_y = np.array(res_iterate)
        plt.plot(interval_x, interval_y)

    plt.scatter(x_nodes, y_nodes, color='red')
    plt.grid()
    plt.show()
```

Результат вызова функции $make\_qubic\_splines\_x(x\_nodes, y\_nodes, x\_tilde)$:

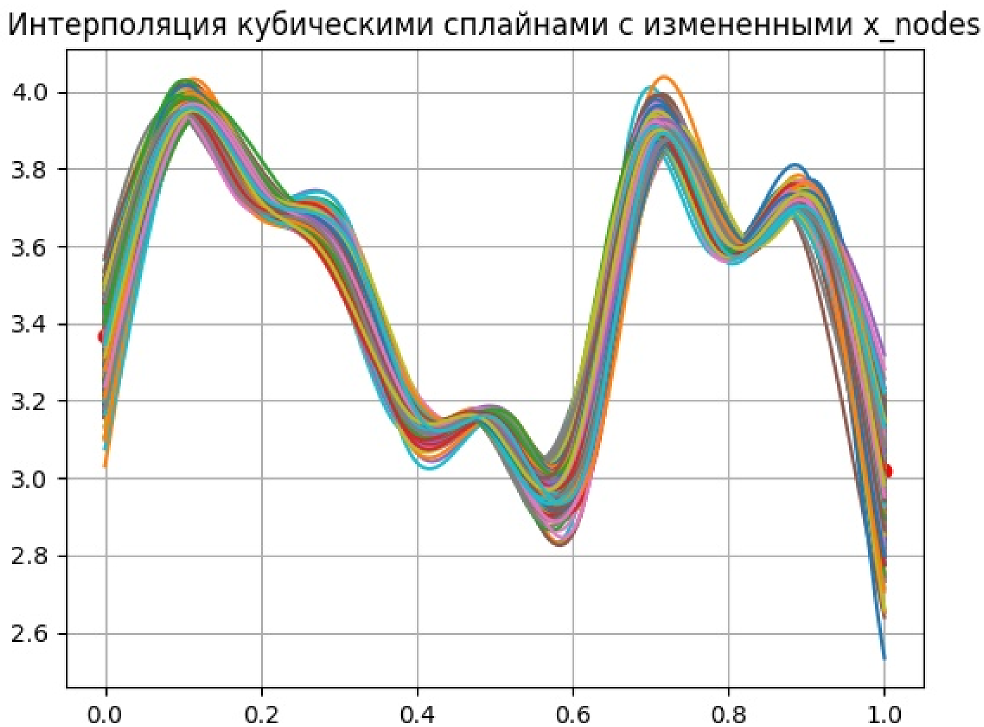


Рис. 7. Графики интерполяции кубическими сплайнами с использованием векторов x_nodes с погрешностью

Построение доверительного интервала для интерполяции кубическими сплайнами с узлами x_tilde, будет производиться по аналогичному алгоритму из 2.3в, г, за исключением того, что теперь для расчета ординат вызывается функция $qubic\_spline$. Разработаем функцию $make\_trust\_interval\_qubic\_x(x\_nodes, y\_nodes, x\_nodes\_res)$:

Листинг функции *make_trust_interval_qubic_x*:

```
def make_trust_interval_qubic_x(x_nodes, y_nodes, x_nodes_res):
    interval_x = np.arange(0.00, 1.0001, 0.01)

    h_l = []
    h_u = []
    median = []

    for x in interval_x:
        temp = []
        for n in range(len(x_nodes_res)):
            x_nodes_i = x_nodes_res[n]
            c = qubic_spline_coeff(x_nodes_i, y_nodes)
```

```
            S = qubic_spline(x, c, x_nodes_i, y_nodes)
            temp.append(S)

        temp = sorted(temp)
        h_l.append(temp[48])
        h_u.append(temp[947])
        median.append(statistics.median(temp))

    plt.plot(interval_x, h_l, color='green', label='h_l(x)')
    plt.plot(interval_x, h_u, color='orange', label='h_u(x)')
    plt.plot(interval_x, median, color='blue', label='median', linewidth='1')
    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Графики h_l(x), h_u(x), усредненный интерполянт, интерполяция
    кубическими \n' + 'сплайнами с использованием x_tilde')
    plt.grid()
    plt.legend()
    plt.show()
```

Результат вызова функции $make\_trust\_interval\_qubic\_x(x\_nodes, y\_nodes, x\_tilde)$:

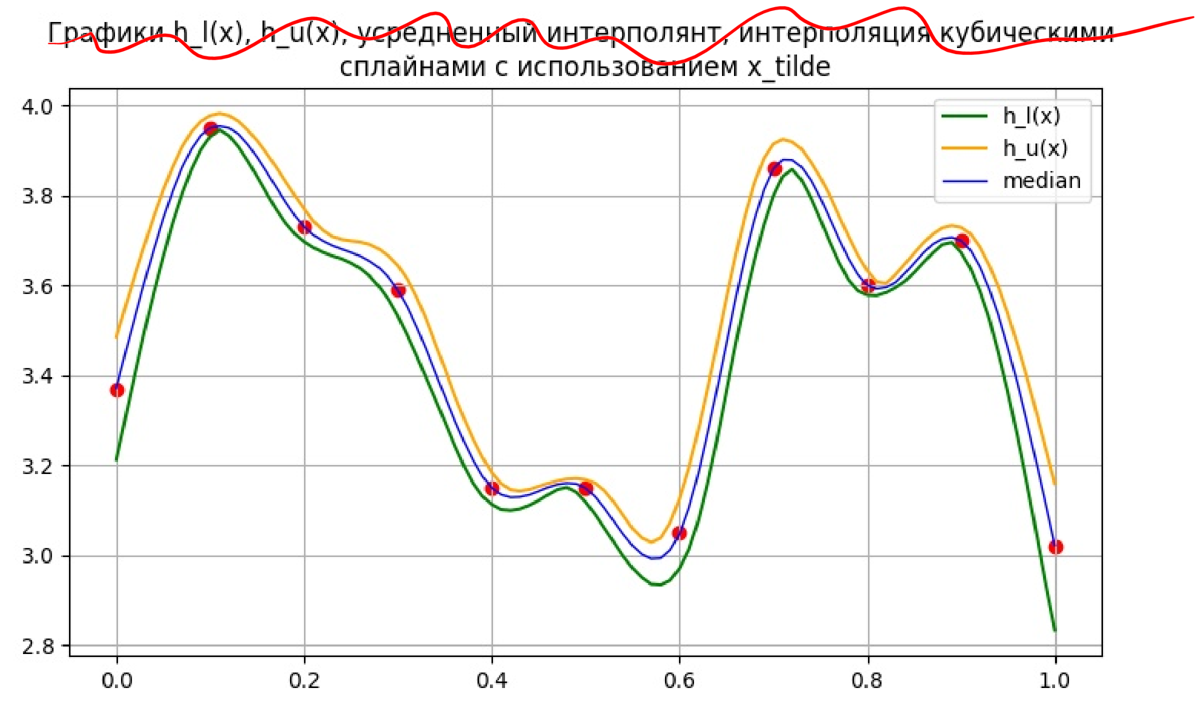


Рис. 8. Графики $\widetilde{h}_l(x), \widetilde{h}_u(x)$ и усредненный интерполянт с узлами x_tilde, интерполяция кубическими сплайнами

Произведем аналогичные построения для кубических спалйанов с узлами x_nodes, y_tilde. Функции будут иметь подобный алгоритм, за исключением того, что передается значение измененных узлов y_tilde. Разработаем функции $make\_qubic\_splines\_y$

$(x\_nodes, y\_nodes, y\_nodes\_res)$ для построения кубических сплайнов со списком y_tilde и $make\_trust\_interval\_qubic\_y(x\_nodes, y\_nodes, y\_nodes\_res)$ для построения доверительного интервала и усредненного интерполянта. Листинги функций приведены ниже.

Листинг функции *make_qubic_splines_y*:

```
def make_qubic_splines_y(x_nodes, y_nodes, y_nodes_res):
    for n in range(len(y_nodes_res)):
        c = qubic_spline_coeff(x_nodes, y_nodes_res[n])
        interval_x = np.arange(x_nodes[0], x_nodes[len(x_nodes) - 1], 0.0005)
        res_iterate = []

        for x in interval_x:
            y_nodes_i = y_nodes_res[n]
            S = qubic_spline(x, c, x_nodes, y_nodes_i)
            res_iterate.append(S)

        interval_y = np.array(res_iterate)
        plt.plot(interval_x, interval_y)

    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Интерполяция кубическими сплайнами с измененными y_nodes')
    plt.grid()
    plt.show()
```

? не нужно пробелов

Результат выполнения функции $make\_qubic\_splines\_y(x\_nodes, y\_nodes, y\_tilde)$:

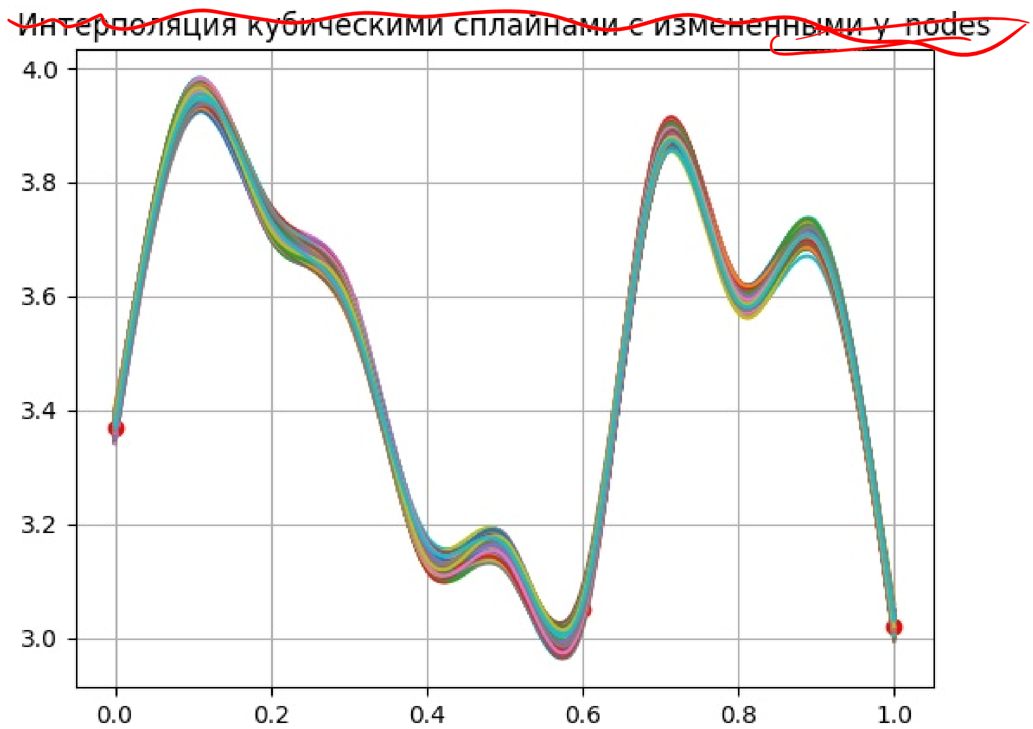


Рис. 9. Графики интерполяции кубическими сплайнами с использованием векторов y_nodes с погрешностью

Листинг функции $\boldsymbol{make\_trust\_interval\_qubic\_y}$**:**

```
def make_trust_interval_qubic_y(x_nodes, y_nodes, y_nodes_res):
    interval_x = np.arange(0.00, 1.0001, 0.01)

    h_l = []
    h_u = []
    median = []

    for x in interval_x:
        temp = []
        for n in range(len(y_nodes_res)):
            y_nodes_i = y_nodes_res[n]
            c = qubic_spline_coeff(x_nodes, y_nodes_i)
            S = qubic_spline(x, c, x_nodes, y_nodes_i)
            temp.append(S)

        temp = sorted(temp)
        h_l.append(temp[48])
```

```
        h_u.append(temp[947])
        median.append(statistics.median(temp))

    plt.plot(interval_x, h_l, color='green', label='h_l(x)')
    plt.plot(interval_x, h_u, color='orange', label='h_u(x)')
    plt.plot(interval_x, median, color='blue', label='median', linewidth='1')
    plt.scatter(x_nodes, y_nodes, color='red')
    plt.title('Графики h_l(x), h_u(x), усредненный интерполянт, интерполяция кубическими \n' + 'сплайнами с использованием y_tilde')
    plt.grid()
    plt.legend()
    plt.show()
```

Результат выполнения функции $make\_trust\_interval\_qubic\_y(x\_nodes, y\_nodes, y\_tilde)$:

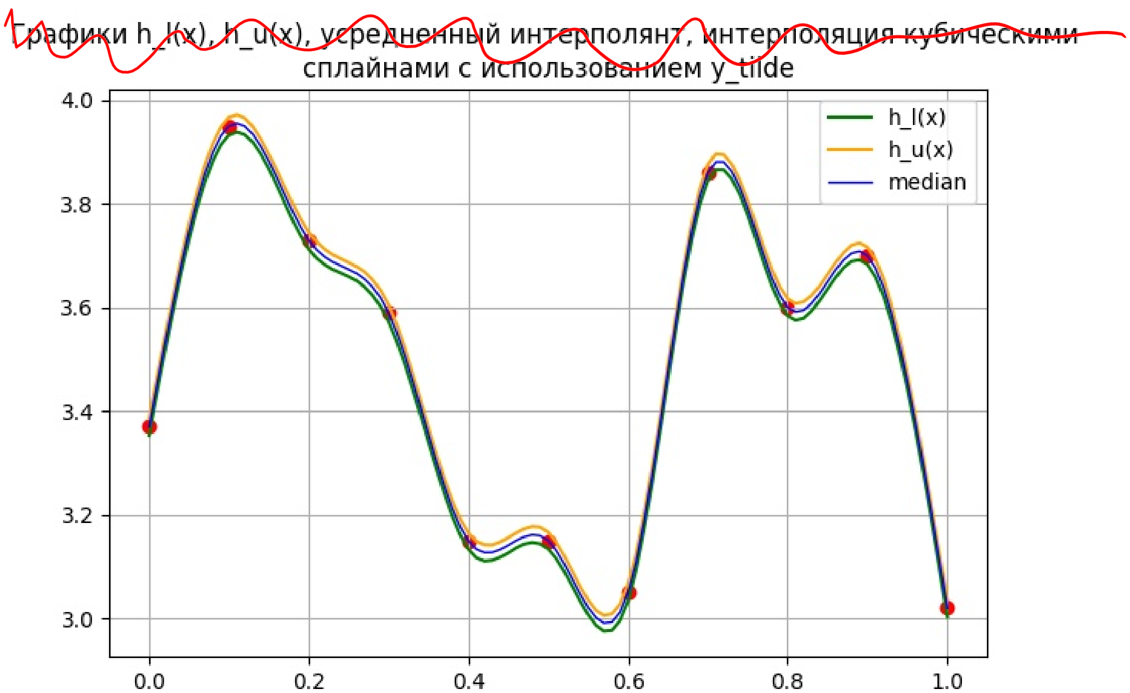


Рис. 10. Графики $\widetilde{h}_l(x), \widetilde{h}_u(x)$ и усредненный интерполянт с узлами y_tilde, интерполяция кубическими сплайнами

Из рис. 7, 8, 9, 10 можно заметить, что при интерполяции кубическими сплайнами результаты получились более точными, если сравнивать с интерполяцией из п. 2.3, 2.4. Это происходит засчет того, что интерполяция кубическими сплайнами предполагает собой локальную интерполяцию полиномом третьей степени, засчет чего отсутствуют осцилляции на концах отрезка, поэтому график гладкий даже с учетом погрешностей, сравнивая результат с глобальной интерполяцией методом Лагранжа из предыдущих пунктов.

Из рис. 8 видно, что доверительный интервал значительно сузился для интерполянтов с учетом погрешности абсцисс и стал приемлемее: ошибка абсцисс в отличие от интерполянтов из п. 2.3 не так сильно влияет на поведение интерполянта, так как из-за нее не появляются паразитные осцилляции.

Из рис. 10 видно, что доверительный интервал для интерполянтов с учетом погрешности ординат также стал незначительно уже и для данного метода также можно утверждать, что ошибка ординаты вносит небольшой вклад в ошибку интерполяции.

Итого, интерполяция методом кубических сплайнов больше подходит для неточных значений, где присутствует погрешность и большого количества точек засчет отсутствия осцилляций.

6 Заключение

1. Интерполяция кубическими сплайнами затрачивает довольно много времени для вычисления, так как использует решение матричного уравнения, однако, этот способ больше подходит для интерполяции с большим количеством узлов или интерполяции, где значения имеют некоторую погрешность засчет того, что полином третьей степени на участках не дает осцилляций.

2. Интерполяция методом Лагранжа полиномом высокой степени порождает осцилляции ближе к концам отрезка, подобный эффект называется феноменом Рунге. При увеличении количества узлов эффект не уменьшится.

3. Интерполяция методом Лагранжа требует меньше времени на построение засчет того, что не использует сложные матричные вычисления, поэтому ее следует использовать для интерполяции с малым количеством узлов.

4. Ошибка абсциссы влияет на погрешность интерполяции. В случае интерполяции методом Лагранжа, погрешность абсциссы заметно усиливает осцилляции на краях отрезка и довольно сильно влияет на конечный результат. В случае интерполяции кубическими сплайнами, погрешность абсциссы заметна, но имеет значительно меньший эффект.

5. Ошибка ординаты слабо влияет на погрешность интерполяции по сравнению с ошибкой абсциссы.

Подводя итог, хотелось бы упомянуть, что для каждой задачи важно выбрать подходящий способ интерполяции, так как неверный выбор может повлечь за собой большие неточности или же быть вычислительно невыгодным по времени.

Список использованных источников

1. Першин А.Ю. Лекции по курсу «Вычислительная математика». Москва, 2018-2021. С. 140.

2. Gaz.Wiki [Электронный ресурс] - Феномен Рунге. Режим доступа: https://gaz.wiki/wiki/ru/Runge%27s_phenomenon

3. Numpy v1.21 Manual [Электронный ресурс]. Режим доступа: https://numpy.org/doc/stable/index.html

4. Matplotlib Documentation [Электронный ресурс]. Режим доступа: https://matplotlib.org/stable/index.html

Выходные данные

Зотов Д. А.. Отчет о выполнении лабораторной работы по дисциплине «Вычислительная математика». [Электронный ресурс] — Москва: 2021. — 32 с. URL: `https://sa2systems.ru:88` (система контроля версий кафедры РК6)

Постановка: © *ассистент кафедры РК-6, PhD А.Ю. Першин*

Решение и вёрстка: © *студент группы РК6-53Б, Зотов Д. А.*

2021, осенний семестр