



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:	Магомедов Зайнутдин Арсланович
Группа:	РК6-54Б
Тип задания:	лабораторная работа
Тема:	Интерполяция в условиях измерений с неопределенностью

Студент

\_\_\_\_\_  
подпись, дата

Магомедов З.А.  
\_\_\_\_\_  
Фамилия, И.О.

Преподаватель

\_\_\_\_\_  
подпись, дата

\_\_\_\_\_  
Фамилия, И.О.

Москва, 2021

# Содержание

<b>Интерполяция в условиях измерений с неопределенностью</b>	<b>3</b>
1    Задание . . . . .	3
2    Цель выполнения лабораторной работы . . . . .	5
3    Выполненные задачи . . . . .	5
3.1    Расчет коэффициентов кубического сплайна по заданным узлам .	6
3.2    Вычисление значения кубического сплайна и его первой производной в заданной точке . . . . .	8
3.3    Построение кубического сплайна . . . . .	9
3.4    Полином Лагранжа. Базисный полином Лагранжа . . . . .	10
3.5    Генерация 1000 векторов и построение интерполянта Лагранжа .	10
3.6    Построение функций $\tilde{h}_l(x)$ и $\tilde{h}_u(x)$ и усредненного интерполянта	12
3.7    Чувствительность участков . . . . .	13
3.8    Повторение с погрешностью в ординатах . . . . .	14
3.9    Повторение кубическим сплайном . . . . .	16
4    Заключение . . . . .	20

# Интерполяция в условиях измерений с неопределенностью

## 1 Задание

Интерполяция, вероятно, является самым простым способом определения недостающих значений некоторой функции при условии, что известны соседние значения. Однако, за кадром зачастую остается вопрос о том, насколько точно мы знаем исходные данные для проведения интерполяции или любой другой аппроксимации. К примеру, исходные данные могут быть получены путем снятия показаний с датчиков, которые всегда обладают определенной погрешностью. В этом случае всегда возникает желание оценить влияние подобных погрешностей и неопределенностей на аппроксимацию. В этом задании на простейшем примере мы познакомимся с интерполяцией в целом (базовая часть) и проанализируем, как неопределенности влияют на ее предсказания (продвинутая часть).

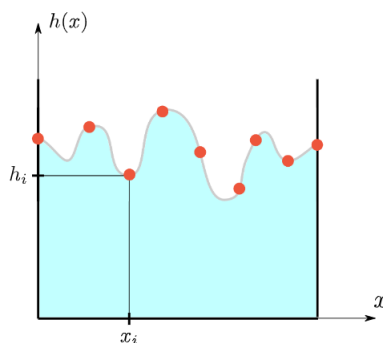


Рис. 1. Поверхность вязкой жидкости (серая кривая), движущейся сквозь некоторую среду (например, пористую). Её значения известны только в нескольких точках (красные узлы).

Таблица 1. Значения уровня поверхности вязкой жидкости (рис. 2)

$x_i$	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

Требуется(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 2) с помощью кубического сплайна и продемонстрировать ее на графике вместе с исходными узлами.

Требуется(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:
  - (a) Сгенерировать 1000 векторов значений  $[\tilde{x}_0, \dots, \tilde{x}_{10}]^T$ , 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 интерполанта 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 интерполанта 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?

## 2 Цель выполнения лабораторной работы

Реализовать интерполяцию кубическими сплайнами и интерполяцию Лагранжа, оценить устойчивость интерполяции с наличием неопределенности (погрешности в исходных данных).

## 3 Выполненные задачи

1. Рассчитаны коэффициенты кубического сплайна по заданным узлам.
2. Написана функция расчета значения сплайна и его первой производной в заданной точке.
3. Построен график зависимости уровня поверхности жидкости от координаты с помощью кубического сплайна.
4. Написана функция, вычисляющая значение  $i$ -го базисного полинома Лагранжа, заданного на узлах, а также функция расчета значения полинома Лагранжа в заданной точке. написана функция, вычисляющая значение  $i$ -го базисного полинома Лагранжа, заданного на узлах, а также функция расчета значения полинома Лагранжа в заданной точке.
5. Сгенерированы 1000 векторов абсцисс узлов с погрешностью, построен интерполянт Лагранжа.
6. Построены 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  $[\tilde{h}_l(x); \tilde{h}_u(x)]$ , будет равна 0.9, а также построить усредненный интерполянт.
7. Проанализирована чувствительность участков интерполянта к погрешностям.
8. Повторение проведенных выше задач с погрешностью в ординатах узлов, тогда как абсциссы узлов известны точно.
9. Повторение задач 6-8 с интерполяцией кубическим сплайном.

### 3.1 Расчет коэффициентов кубического сплайна по заданным узлам

Для функции  $f(x)$ , заданной в  $n$  интерполяционных узлах  $a = x_1, x_2, \dots, x_n = b$  на отрезке  $[a; b]$ , можно построить единственный кубический сплайн  $S(x)$ , для которого верно:

1.  $S(x)$  кусочно задана кубическими многочленами  $S_i(x)$  на каждом отрезке  $[x_i; x_{i+1}]$ , где  $i = 1, \dots, n-1$ ;
2.  $S_i(x_i) = f(x_i)$  и  $S_i(x_{i+1}) = f(x_{i+1})$ ,  $i = 1, \dots, n-1$ ;
3. Значения смежных многочленов совпадают в общих узлах:  
 $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ ,  $i = 1, \dots, n-2$ ;
4. Значения первых производных смежных многочленов совпадают в общих узлах:  
 $S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$ ,  $i = 1, \dots, n-2$ ;
5. Значения вторых производных смежных многочленов совпадают в общих узлах:  
 $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})$ ,  $i = 1, \dots, n-2$ ;
6. Заданы граничные условия:
  - Естественные граничные условия:  $S''(x_1) = S''(x_n) = 0$
  - Граничные условия на касательную:  $S'(x_1) = f'(x_1)$  и  $S'(x_n) = f'(x_n)$

Поскольку кубический многочлен задается 4 константами, то для задания кубического сплайна нам необходимо определить  $4(n-1)$  констант. Запишем кубический многочлен  $S_i(x)$  на отрезке  $[x_i; x_{i+1}]$  в форме

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

Это нам автоматически дает, что  $S_i(x_i) = a_i = f(x_i)$ . То есть,  $a_i$  равно ординате соответствующего узла  $y_i$

Затем мы находим вектор  $c$ , решив матричное уравнение  $Ac = b$  (**Важно!** Не путать вектор  $b$  в матричном уравнении и коэффициент кубического сплайна. Это разные вектора)

$$h_i = x_{i+1} - x_i$$

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

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

Получив вектор  $c$ , получим вектор  $b$ , а затем вектор  $d$ :

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

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

В конечном итоге мы получим матрицу коэффициентов размерностью  $(N - 1) \times 3$ , где  $N$  - количество узлов интерполяции.

Ниже приведен фрагмент кода на языке Python, вычисляющий матрицу коэффициентов.

---

```

1 def cubic_spline_coeff(x_nodes, y_nodes):
2     a = y_nodes
3     h = [x_nodes[i + 1] - x_nodes[i] for i in range(0, 10)]
4     res = [0 for i in range(0, 11)]
5     for i in range(1, 10):
6         res[i] = 3*(a[i+1]-a[i])/h[i] - 3*(a[i]-a[i-1])/h[i-1]
7     matrix = np.zeros((11, 11))
8     for i in range(0, 11):
9         for j in range(0, 11):
10             if (i == j):
11                 if (i > 0) and (i < 10):
12                     matrix[i, j] = (h[i]+h[i-1])*2
13                 else:
14                     matrix[i, j] = 1
15             if (j == i - 1) and (i < 10):
16                 matrix[i, j] = h[j]
17             if (j == i + 1) and (i > 0):
18                 matrix[i, j] = h[i]
19     # print(res)
20     matrix = LA.inv(matrix)
```

```

21 c = matrix @ res
22 b = [(a[i+1]-a[i])/h[i] - h[i]*(c[i+1]+2*c[i])/3 for i in range(0,10)]
23 d = [(c[i+1]-c[i])/(3*h[i]) for i in range(0,10)]
24 c = np.delete(c,(10), axis = 0)
25 qs_coeff = np.c_[b, c, d]
26 return(qs_coeff)

```

---

Изначально мы находим вектор  $a$ , который равен вектору ординат узлов интерполяции. Затем формируем вектор  $h$ , а также вектор  $res = b$  (не путать с вектором коэффициентов, это вектор из матричного уравнения). Затем мы формируем матрицу  $matrix$ . Воспользовавшись функцией  $LA.inv(matrix)$  библиотеки *numpy*, находим обратную матрицу и перемножаем с вектором  $res$ . Оттуда находим вектор коэффициентов  $c$ . Находим вектора  $b$  и  $d$  по формулам. Затем удаляем последний элемент вектора  $c$ , поскольку он не играет роли, так как он для последнего узла, а там уже нет кубического многочлена. Затем с помощью функции  $np.c_[b, c, d]$  библиотеки *numpy* конкатенируем вектора в матрицу коэффициентов. Затем возвращаем её.

### 3.2 Вычисление значения кубического сплайна и его первой производной в заданной точке

Сначала запишем уравнение кубического многочлена в общем виде:

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

По условию нам необходимо посчитать значение функции сплайна и его производной в заданной точке. При этом нам необходимо проверить, входит ли точка в отрезок  $[0; 1]$ . Мы определяем узлы, между которыми расположена заданная точка. А затем все подставляем в уравнение сплайна.

Для этого я написал отдельную функцию, которая возвращает индекс нужного узла в зависимости от заданной точки.

Код этой функции приведен ниже:

---

```

1 def correct_index(x, x_nodes):
2     n = len(x_nodes)
3     if x > x_nodes[n - 1]:
4         return n - 2
5     if x < x_nodes[0]:
6         return 0
7
8     for i in range(0, n - 1):
9         if x >= x_nodes[i] and x <= x_nodes[i + 1]:
10            return i

```

---

Далее были реализованы функции собственно для подсчета значения сплайна и его производной в заданной точке.



Код приведен ниже:

---

```
1 def qubic_spline(x, x_nodes, y_nodes, qs_coeff):
2     node = x_nodes[correct_index(x, x_nodes)]
3     i = correct_index(x, x_nodes)
4     S = y_nodes[i] + qs_coeff[i][0]*(x-x_nodes[i]) + qs_coeff[i][1]*((x-x_nodes[i])**2) +
        qs_coeff[i][2]*((x-x_nodes[i])**3)
5     return S
```

---

```
1 def d_qubic_spline(x, x_nodes, qs_coeff):
2     node = x_nodes[correct_index(x, x_nodes)]
3     i = correct_index(x, x_nodes)
4     _S = qs_coeff[i][0] + 2*qs_coeff[i][1]*(x-x_nodes[i]) +
        3*qs_coeff[i][2]*((x-x_nodes[i])**2)
5     return(_S)
```

---

### 3.3 Построение кубического сплайна

Для построения кубического сплайна мы сформировали матрицу коэффициентов. Для построения сплайна генерируется тысяча точек функцией *linspace()* библиотеки *numpy*. Затем в каждой из этих точек вычисляем значение сплайна, используя написанную ранее функцию *qubic\_spline()*. Затем, используя библиотеку *matplotlib*, рисуем график:

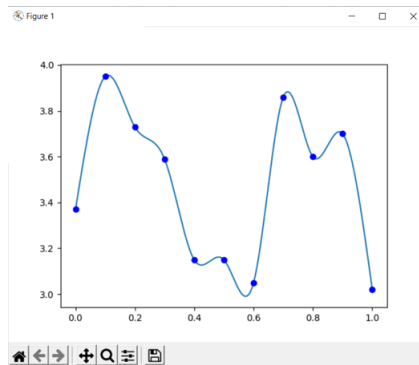


Рис. 2. Кубический сплайн

Код функции приведен ниже:

---

```

1 def graphic_cubic_spline(x_nodes, y_nodes, qs_coeff):
2     _x = np.linspace(0, 1, 1000)
3     F = [cubic_spline(_x[i], x_nodes, y_nodes, qs_coeff) for i in range(0, 1000)]
4     plt.plot(_x, F)
5     plt.plot(x_nodes, y_nodes, 'o', color='blue')
6     plt.show()

```

---

### 3.4 Полином Лагранжа. Базисный полином Лагранжа

Пусть функция  $f(x)$  задана в  $n$  интерполяционных узлах  $a = x_1, x_2, \dots, x_n = b$  на отрезке  $[a; b]$ . Тогда интерполяционным многочленом для функции  $f(x)$  и соответствующих узлов интерполяции называется функция

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

где  $l_i(x) = \prod_{i \neq j} \frac{x - x_j}{x_i - x_j}$  является базисным многочленом  $(n - 1)$  - й степени

Код функций представлен ниже:

---

```

1 def l_i(i, x, x_nodes):
2     l = 1
3     for j in range(0, 11):
4         if (i != j):
5             l = l * (x - x_nodes[j]) / (x_nodes[i] - x_nodes[j])
6
7     #print(l)
8     return l
9
10 def L(x, x_nodes, y_nodes):
11     L = 0.0
12     for i in range(0, 11):
13         L = L + y_nodes[i] * l_i(i, x, x_nodes)
14
15     #print(L)
16     return L

```

---

### 3.5 Генерация 1000 векторов и построение интерполянта Лагранжа

С помощью функции *random.normal()* библиотеки *numpy* генерирую число  $Z$ , которое является случайной величиной с нормальным распределением с нулевым математическим ожиданием и стандартным отклонением  $10^{-2}$ . Сгенерировав  $Z$ , генерируем 1000 векторов абсцисс узлов.

Код функции генерации векторов представлен ниже:

```
1 def gen_vector(x_nodes):
2     X_vector = []
3     for i in range(0, len(x_nodes)):
4         Z = np.random.normal(loc = 0.0, scale=0.01)
5         X_vector.append(x_nodes[i] + Z)
6     # print(X_vector)
7     return(X_vector)
```

Затем, получив вектора абсцисс узлов, генерируем 1000 точек для построения каждого из 1000 интерполянтов. Таким образом итерационно 1000 раз получаю вектор абсцисс узлов и 1000 точек для построения и вызываю функцию подсчета значения полинома Лагранжа в каждой из точек. Далее формирую для каждого интерполянта график с помощью функции *plt.plot()* библиотеки *matplotlib*. Сформировав так все графики, вывожу их на экран:

Код функции представлен ниже:

```
1 def graph_thousand_vectors(x_nodes, y_nodes):
2     for i in range(0, 1000):
3         x_shift = np.linspace(0, 1, 1000)
4         #print(x_shift)
5         x_nodes_new = gen_vector(x_nodes)
6         Function = [L(x_shift[i], x_nodes_new, y_nodes) for i in range(0, 1000)]
7         #print(x_shift, Function)
8         plt.plot(x_shift, Function)
9     plt.grid(True)
10    plt.show()
```

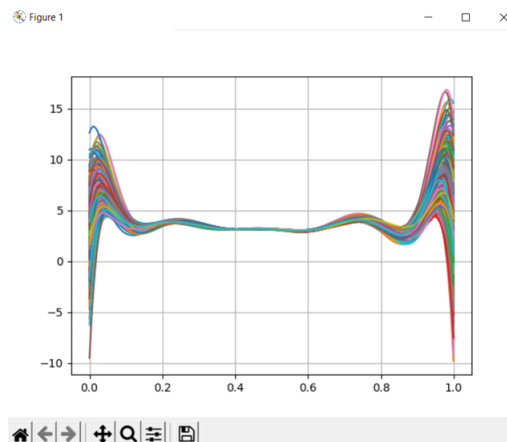


Рис. 3. Интерполяция полиномом Лагранжа с погрешностью в абсциссах узлов (1000 шт.)

### 3.6 Построение функций $\tilde{h}_l(x)$ и $\tilde{h}_u(x)$ и усредненного интерполянта

По условию нам необходимо построить такие функции  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$ , что вероятность того, что значение в точке  $x \in [0; 1]$  будет лежать в интервале  $[\tilde{h}_l(x); \tilde{h}_u(x)]$  будет равна 0.9.

Прежде всего следует ввести понятие доверительного интервала. Доверительным интервалом называется интервал, вычисленный по выборочным данным, который с заданной вероятностью (доверительной) накрывает неизвестное истинное значение параметра распределения.

В нашем случае доверительным интервалом будет участок плоскости, образуемый графиками функций  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  при  $x \in [0; 1]$ . Внутри него с вероятностью 0.9 по условию должно быть значение в точке  $x$ .

Доверительный интервал был найден следующим образом. Изначально я формирую 1000 точек для построения. Затем итерационно 1000 раз генерирую абсциссы узлов и, соответственно, значения полинома Лагранжа в этих точках. Все это я записываю в матрицу. (Я получаю матрицу 1000 на 1000). То есть, для каждой точки  $x$ , сгенерированной для построения, я генерирую тысячу значений  $y$ .

Таким образом, столбец матрицы есть 1000 значений  $y$  для одной точки  $x$ . Затем каждый столбец сортирую по возрастанию. И для построения функций выбираю 2 точки таким образом, что 900 из 1000 полученных точек находятся между этими двумя. Так я и получаю вероятность, равную 0.9.

То есть, если предположить, что столбец значений, скажем, называется *list*, то для каждой  $i$ -ой точки

$$\tilde{h}_l[i] = \text{list}[49]$$

$$\tilde{h}_u[i] = \text{list}[949]$$

При этом усредненный интерполянт находится посередине, то есть количество точек будет поровну между верхним и усредненным и усредненным и нижним интерполянтами соответственно. То есть,

$$\tilde{h}_{\text{medium}}[i] = \text{list}[499]$$

Код функции представлен ниже:

---

```
1 def graph_h_functions_lagrange(x_nodes, y_nodes):
2     h_l = []
3     h_medium = []
4     h_u = []
5     matrix = []
6     x = np.linspace(0, 1, 1000)
7     for i in range(0, 1000):
8         y = []
```

```

9     gen_nodes = gen_vector(x_nodes)
10    for j in range(0, 1000):
11        y.append(L(x[j], gen_nodes, y_nodes))
12        matrix.append(y)
13    # print(matrix)
14
15    for i in range(0, 1000):
16        list = []
17        for j in range(0, 1000):
18            list.append(matrix[j][i])
19        list = sorted(list)
20        h_l.append(list[49])
21        h_medium.append(list[499])
22        h_u.append(list[949])
23
24    plt.plot(x, h_l)
25    plt.plot(x, h_medium)
26    plt.plot(x, h_u)
27    plt.fill_between(x, h_u, h_l, color = 'yellow')
28    plt.show()

```

График функций и усредненного интерполянта представлен ниже, закрасил доверительный интервал для наглядности:

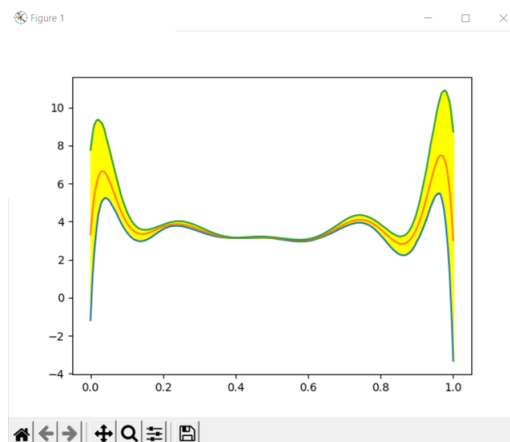


Рис. 4. Доверительный интервал, усредненный интерполянт и функции  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  с погрешностью в абсциссах(Лагранж)

### 3.7 Чувствительность участков

Несложно заметить, что концы интервала наиболее чувствительны к погрешностям. Объясняется это тем, что при больших степенях многочлена возникают большие осцилляции. Все дело в том, что локальные экстремумы расположены близко к концам,

а именно там значения базисного полинома сильнее всего колеблются, т.е. происходят довольно сильные осцилляции. Такой эффект осцилляций называется феноменом Рунге. Стоит сказать, что у самого полинома Лагранжа есть «проблемы» при увеличении степени. Например, 3 степень полинома Лагранжа сильнее колеблется, нежели тот же кубический сплайн. В роли «минимизатора» осцилляций могут выступать узлы Чебышева (выбор специальной сетки для интерполяции).

### 3.8 Повторение с погрешностью в ординатах

С помощью функции `random.normal()` библиотеки `numpy` генерирую число  $Z$ , которое является случайной величиной с нормальным распределением с нулевым математическим ожиданием и стандартным отклонением  $10^{-2}$ . Сгенерировав  $Z$ , генерируем 1000 векторов абсцисс ординат. Функция, генерирующая такие вектора, представлена выше (смотрите п.5)

Далее все продельвается аналогичным образом, как в п.5, только здесь генерируются ординаты узлов.

Код функции приведен ниже:

---

```

1 def graph_thousand_vectors(x_nodes, y_nodes):
2     for i in range(0, 1000):
3         x_shift = np.linspace(0, 1, 1000)
4         #print(x_shift)
5         y_nodes_new = gen_vector(y_nodes)
6         Function = [L(x_shift[i], x_nodes, y_nodes_new) for i in range(0, 1000)]
7         #print(x_shift, Function)
8         plt.plot(x_shift, Function)
9     plt.grid(True)
10    plt.show()

```

---

График представлен ниже:

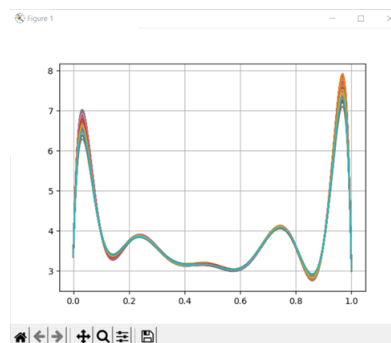


Рис. 5. Интерполяция полиномом Лагранжа с погрешностью в ординатах узлов (1000 шт.)

Далее, что касается построения функций  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  и их усредненного интерполянта, тут происходит аналогично п.6 только с погрешностями в ординатах узлов. Здесь уже генерируются итерационно новые ординаты узлов 1000 раз, а далее для

каждого сгенерированного вектора ординат узлов генерируется 1000 значений полинома Лагранжа. Таким образом, мы формируем матрицу аналогично п.6. А далее все происходит, как в п.6(смотрите пункт 6)

Код функции представлен ниже:

---

```
1 def graph_h_functions_lagrange(x_nodes, y_nodes):
2     h_l = []
3     h_medium = []
4     h_u = []
5     matrix = []
6     x = np.linspace(0, 1, 1000)
7     for i in range(0, 1000):
8         y = []
9         gen_nodes = gen_vector(y_nodes)
10        for j in range(0, 1000):
11            y.append(L(x[j], x_nodes, gen_nodes))
12        matrix.append(y)
13    # print(matrix)
14
15    for i in range(0, 1000):
16        list = []
17        for j in range(0, 1000):
18            list.append(matrix[j][i])
19        list = sorted(list)
20        h_l.append(list[49])
21        h_medium.append(list[499])
22        h_u.append(list[949])
23
24    plt.plot(x, h_l)
25    plt.plot(x, h_medium)
26    plt.plot(x, h_u)
27    plt.fill_between(x, h_u, h_l, color = 'yellow')
28    plt.show()
```

---

График представлен ниже:

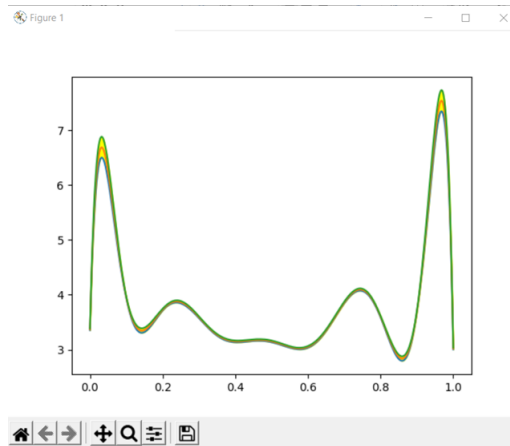


Рис. 6. Доверительный интервал, усредненный интерполянт и функции  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  с погрешностью в ординатах(Лагранж)

Таким образом, повторив эксперимент уже с погрешностью в ординатах узлов, можно заметить, что доверительный интервал заметно сократился. Можно сказать, что небольшое изменение в абсциссе сильно меняет результат вычисления значения базисного полинома Лагранжа, при этом ордината узла при вычислении значения полинома Лагранжа «выступает в 1-ой степени». То есть, небольшое изменение абсциссы приводит к большим по сравнению с изменением абсциссы осцилляциям, чего нельзя сказать об ординате узлов.

### 3.9 Повторение кубическим сплайном

В этом пункте все происходит аналогично пунктам 5,6 и 8, за исключением того, что вызывается функция вычисления значения сплайна в точке(напомню, что в указанных выше пунктах вызывается функция вычисления значения полинома Лагранжа в точке)

Для погрешности в абсциссах:

---

```

1 def graph_qubic_spline(x_new):
2     x_nodes = [i / 10 for i in range(0, 11)]
3     x_new_nodes = gen_vector(x_nodes)
4     _y = []
5     for i in range(0, len(x_new)):
6         F = qubic_spline(x_new[i], x_new_nodes, y_nodes,
7                           qubic_spline_coeff(x_new_nodes, y_nodes))
7         _y.append(F)
8     return _y
9
10 def graph_h_functions_qubic(x_nodes, y_nodes):
11     h_l = []
12     h_medium = []

```



```

13  h_u = []
14  matrix = []
15  x = np.linspace(0, 1, 1000)
16  for i in range(0, 1000):
17      y = graph_qubic_spline(x)
18      matrix.append(y)
19
20  for i in range(0, 1000):
21      list = []
22      for j in range(0, 1000):
23          list.append(matrix[j][i])
24      list = sorted(list)
25      h_l.append(list[49])
26      h_medium.append(list[499])
27      h_u.append(list[949])
28
29  plt.plot(x, h_l)
30  plt.plot(x, h_medium)
31  plt.plot(x, h_u)
32  plt.fill_between(x, h_u, h_l, color = 'yellow')
33  plt.show()

```

График представлен ниже:

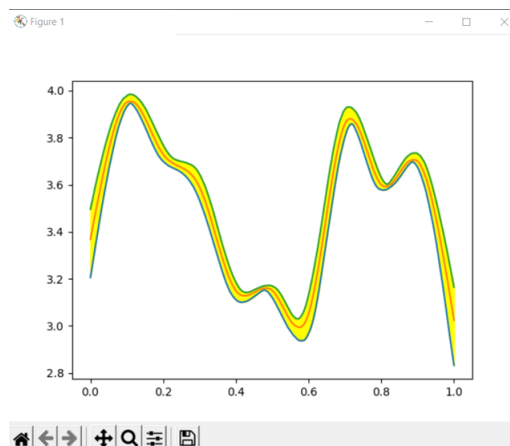


Рис. 7. Доверительный интервал, усредненный интерполянт и функции  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  с погрешностью в абсциссах (Кубический сплайн)

Для погрешности в ординатах:

---

```
1 def graph_qubic_spline(x_new):
2     x_nodes = [i / 10 for i in range(0, 11)]
3     y_new_nodes = gen_vector(y_nodes)
4     _y = []
5     for i in range(0, len(x_new)):
6         F = qubic_spline(x_new[i], x_nodes, y_new_nodes, qubic_spline_coeff(x_nodes,
7             y_new_nodes))
8         _y.append(F)
9     return _y
10
11 def graph_h_functions_qubic(x_nodes, y_nodes):
12     h_l = []
13     h_medium = []
14     h_u = []
15     matrix = []
16     x = np.linspace(0, 1, 1000)
17     for i in range(0, 1000):
18         y = graph_qubic_spline(x)
19         matrix.append(y)
20
21     for i in range(0, 1000):
22         list = []
23         for j in range(0, 1000):
24             list.append(matrix[j][i])
25         list = sorted(list)
26         h_l.append(list[49])
27         h_medium.append(list[499])
28         h_u.append(list[949])
29
30     plt.plot(x, h_l)
31     plt.plot(x, h_medium)
32     plt.plot(x, h_u)
33     plt.fill_between(x, h_u, h_l, color = 'yellow')
34     plt.show()
```

---

График представлен ниже:

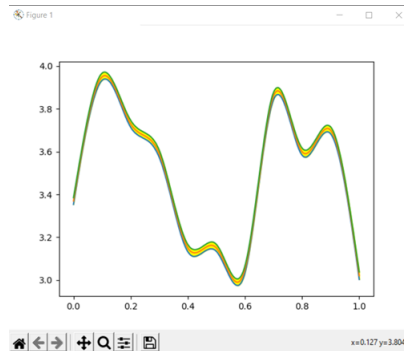


Рис. 8. Доверительный интервал, усредненный интерполянт и функции  $\tilde{h}_l(x)$  и  $\tilde{h}_u(x)$  с погрешностью в ординатах (Кубический сплайн)

Для погрешности в абсциссах:

---

```

1 def graph_thousand_vectors(x_nodes, y_nodes):
2     for i in range(0, 1000):
3         x_shift = np.linspace(0, 1, 1000)
4         #print(x_shift)
5         x_nodes_new = gen_vector(x_nodes)
6         Function = [qubic_spline(x_shift[i], x_nodes_new, y_nodes,
7                               qubic_spline_coeff(x_nodes_new, y_nodes)) for i in range(0, 1000)]
8         #print(x_shift, Function)
9         plt.plot(x_shift, Function)
10    plt.grid(True)
    plt.show()

```

---

График представлен ниже:

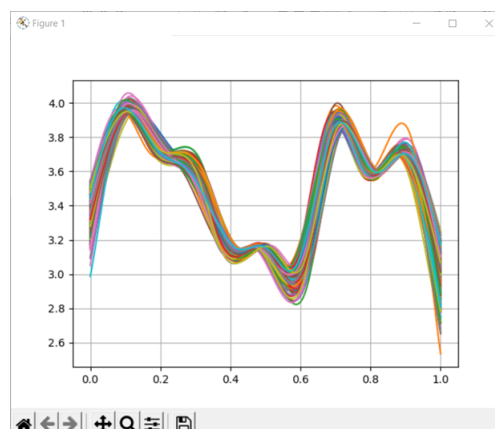


Рис. 9. Интерполяция кубическим сплайном с погрешностью в абсциссах узлов (1000 шт.)

Для погрешности в ординатах:

---

```

1 def graph_thousand_vectors(x_nodes, y_nodes):

```

```

2   for i in range(0, 1000):
3       x_shift = np.linspace(0, 1, 1000)
4       #print(x_shift)
5       y_nodes_new = gen_vector(y_nodes)
6       Function = [qubic_spline(x_shift[i], x_nodes, y_nodes_new,
7                               qubic_spline_coeff(x_nodes, y_nodes_new)) for i in range(0, 1000)]
7       #print(x_shift, Function)
8       plt.plot(x_shift, Function)
9   plt.grid(True)
10  plt.show()

```

График представлен ниже:

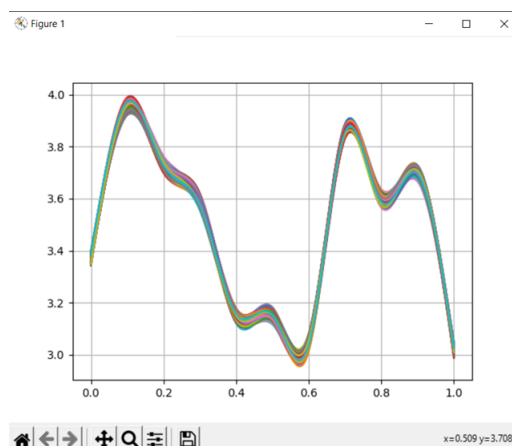


Рис. 10. Интерполяция кубическим сплайном с погрешностью в ординатах узлов (1000 шт.)

Таким образом, можно сделать вывод, что интерполяция кубическим сплайном более точна, чем полиномом Лагранжа. Дело в том, что при интерполяции Лагранжа начинаются проблемы при увеличении степени, в т.ч. при 3-й степени, появляются довольно большие осцилляции, чего не скажешь об интерполяции кубическими сплайнами.

## 4 Заключение

В процессе выполнения лабораторной работы были изучены, применены, реализованы алгоритмы для интерполяции Лагранжа и интерполяции кубическими сплайнами. Мы узнали, что интерполяция кубическими сплайнами более точная нежели интерполяция многочленом Лагранжа. При интерполяции многочленом Лагранжа возникают довольно большие осцилляции. Посмотрев на график с доверительным интервалом, видим, что наибольшая ширина доверительного интервала на краях отрезка  $[0; 1]$ , а при погрешности в ординате наибольшей ширины интервал достигает в экстремумах. А при интерполяции кубическими сплайнами ширина интервала, можно сказать,



постоянна, меняется незначительно. Мы поняли, что небольшой сдвиг по оси абсцисс(погрешность) оказывает много большее влияние на результирующую функцию, нежели сдвиг по оси ординат.

### Список использованных источников

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

### Выходные данные

Магомедов З.А.. Отчет о выполнении лабораторной работы по дисциплине «Вычислительная математика». [Электронный ресурс] — Москва: 2021. — 21 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 РК6-54Б, Магомедов З.А.

2021, осенний семестр