TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Углубленный курс информатики

Лекция 8

Обработка экспериментальных данных

Вячеслав Алексеевич Чузлов
к.т.н., доцент ОХИ ИШПР

25 марта 2023 г.

TOMSK POLYTECHNIC UNIVERSITY
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

СОДЕРЖАНИЕ

Введение

- При изучении химических и химико-технологических процессов очень часто возникает необходимость в обработке и анализе данных, полученных экспериментально, с применением результатов этой обработки в последствии при моделировании и проектировании реальных процессов.

- Предположим функция выхода целевого продукта реакции от концентрации исходного компонента записана как $y = f(x)$. Таким образом, любому значению концентрации исходного компонента x соответствует значение выхода целевого продукта y.

- В действительности решение функции $f(x)$ является сложной задачей, требующей проведения дорогостоящего эксперимента или применения затратных аналитических методов количественного определения концентрации компонента. В дополнение к этому, аналитическое выражение функции $f(x)$ может быть также затруднено или вообще невозможно.

- В таких случаях составляют небольшую таблицу значений выходного параметра от аргумента строят по ней функцию $y = \tilde{f}(x)$, где x – концентрация исходного компонента, y – концентрация целевого продукта, $\tilde{f}$ – приближенная функция, описывающая данные эксперимента.

TOMSK
POLYTECHNIC
UNIVERSITY

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Интерполяция

Интерполяция

При выполнении инженерных расчетов в области химии и химической технологии очень распространена задача установления функции $f(x)$ для всех значений x на отрезке $[a, b]$ при известных значениях в некотором конечном числе точек из этого отрезка. Одним из способом решения данной задачи является интерполяция.

Задача интерполяции в работе химика-технолога может возникнуть в следующих случаях:

- при работе с табличными данными;
- при обработке экспериментальных табличных данных для установления функциональной взаимосвязи;
- при необходимости упрощения сложной с точки зрения вычислений функции более простой зависимостью;
- при дифференцировании и интегрировании.

Постановка задачи

Предположим, что на отрезке $[x_0, x_n]$ заданы $n+1$ точки $x_0, x_1, x_2, \dots x_n$, которые называют узлами интерполяции, и значения некоторой интерполируемой функции $y = f(x)$ в этих точках интерполяции. Иными словами, задана таблица экспериментальных значений функции $y = f(x)$:

X	x_0	x_1	$\dots$	x_n
Y	$y_0 = f(x_0)$	$y_1 = f(x_1)$	$\dots$	$y_n = f(x_n)$

Необходимо найти значения этой функции для промежуточных значений аргумента, которых нет в таблице, но которые принадлежат отрезку $[x_0, x_n]$. Аналитическое выражение функции $y = f(x)$ по таблице значений, как правило, невозможно, поэтому вместо нее строят другую функцию, более легкую в вычислениях и имеющую такую же таблицу значений, что и $f(x)$:

$$\begin{aligned} P_m(x_0) &= f(x_0) = y_0 \\ &\dots \\ P_m(x_i) &= f(x_i) = y_i \end{aligned} \qquad (1)$$

где $i = 0, 1, 2, \dots, n$.

Точки x_i называются **узлами интерполяции**, функция $f(x)$ называется **интерполируемой функцией**, а полином $P_m(x)$ – **интерполяционным полиномом**. Задача интерполяции сводится к нахождению приближенных значений табличной функции при аргументах x, не совпадающих с узловыми. В том случае, если значение аргумента x расположено между узлами $x_0 \leqslant x \leqslant x_n$, то нахождение приближенного значения функции $f(x)$ называется **интерполяцией**; если интерполирующую функцию вычисляют за пределами интервала $[x_0, x_n]$, то процесс называют **экстраполяцией**.

Постановка задачи

Графическая задача интерполяции заключается в построении такой интерполирующей функции, график которой будет проходить через все точки исходных данных (рисунок 1).

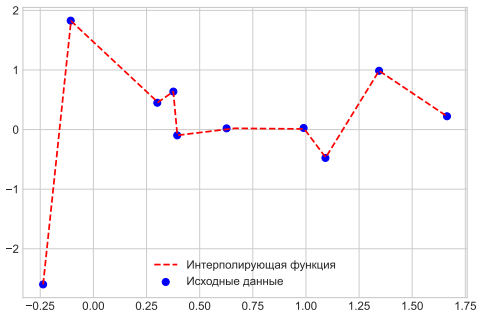

Рисунок 1 – Графическая задача интерполяции

Близость интерполяционного полинома к интерполируемой функции обеспечивается тем, что их значения совпадают в узлах интерполяции. При решении задач интерполяции принимаются следующие положения:

- интерполируемая функция непрерывна на отрезке $[a, b]$ и в каждой точке имеет конечные производные любого порядка;
- узлы интерполирования неравны друг другу.

Линейная интерполяция

Наиболее простым и часто используемым видом интерполяции является линейная интерполяция. Идея линейной интерполяции состоит в том, что заданные точки x_i, y_i при $i = 0, 1, 2, \ldots, n$ соединяются прямыми отрезками и функция $f(x)$ заменяется ломаной с вершинами в данных точках.

Уравнения каждого из отрезков в общем случае разные. Так как имеется n интервалов ($x_i - 1, x_i$), то для каждого из них в качестве уравнения интерполяционного полинома используется уравнение прямой, которая проходит через две точки. Таким образом, для i-го интервала можно написать уравнение прямой, проходящей через точки (x_{i-1}, y_{i-1}) и (x_i, y_i), в следующем виде:

$$\frac{y - y_{i-1}}{y_i - y_{i-1}} = \frac{x - x_{i-1}}{x_i - x_{i-1}}$$

откуда

$$y = a_i x + b_i,\ x_{i-1} \leqslant x \leqslant x_i$$
$$a_i = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \qquad (2)$$
$$b_i = y_{i-1} - a_i x_{i-1}$$

Таким образом, при использовании линейной интерполяции сначала необходимо определиться с интервалом, в который попадает значение аргумента x, после чего нужно подставить его в формулу (2), чтобы найти приближенное значение функции в этой точке.

Интерполяционный полином Лагранжа

Рассмотрим таблично заданную функцию. К примеру, это могут быть значения теплоемкости вещества или значения концентраций продуктов реакции при различной температуре, полученные при проведении лабораторного эксперимента.

x	x_0	x_1	$\dots$	x_n
f(x)	$f(x_0)$	$f(x_1)$	$\dots$	$f(x_n)$

Будем считать, что в общем случае шаг таблицы неравномерный и значения $x_0, x_1, \dots, x_n$ не являются равноотстоящими.
Искомый интерполяционный полином на отрезке $[x_0, x_n]$ запишем следующим образом:

$$P_m(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m \tag{3}$$

С геометрической точки зрения задача сводится к построению кривой, проходящей через заданные точки. С аналитической точки зрения задача сводится к решению системы уравнений:

$$\mathrm{Y}_i(x) = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_m x_i^m,\ i = 0 \dots n \tag{4}$$

Для того, чтобы определить коэффициенты полинома (3), требуется располагать $n+1$ узлами интерполяции. А чтобы система имела единственное решение, требуется, чтобы количество неизвестных коэффициентов полинома $(m+1)$ было равно количеству уравнений n, иначе говоря, должно выполняться равенство $m = n - 1$.

Интерполяционный полином Лагранжа

Допустим, что в $(n+1)$ точках $x_0, x_1, \ldots, x_n$ определены значения $y_0, y_1, \ldots, y_n$. Необходимо построить полином $P_n(x)$, имеющий в узлах интерполяции заданные значения y_i:

$$P_n(x_i) = y_i,\ i = 0, 1, \ldots, n$$

Лагранжем была предложена следующая форма интерполяционного полинома:

$$P_n(x) = \sum_{i=0}^{n} Y_i \cdot L_i(x) \tag{5}$$

где $L_i(x)$ – множитель Лагранжа, который записывается следующим образом:

$$L_i(x) = \frac{(x - x_0)\ldots(x - x_{i-1})\,(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_0)\ldots(x_i - x_{i-1})\,(x_i - x_{i+1})\ldots(x_i - x_n)} = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)} \tag{6}$$

Таким образом, полином Лагранжа можно записать следующим образом:

$$P_n(x) = \sum_{i=0}^{n} y_i \left(\prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{(x - x_k)}{(x_i - x_k)} \right) \tag{7}$$

Интерполяционный полином Лагранжа

В развернутом виде полином Лагранжа можно представить следующим образом:

$$P_n(x) = Y_0 \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)\dots(x_0-x_n)} + \\ + Y_1 \frac{(x-x_0)(x-x_2)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)\dots(x_1-x_n)} + \\ + \dots + \\ + Y_n \frac{(x-x_0)(x-x_1)(x-x_2)\dots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)(x_n-x_2)\dots(x_n-x_{n-1})} \quad (8)$$

Пример использования интерполяционного полинома Лагранжа

Дана таблица значений теплоемкости вещества при различной температуре ($C_p = f(T)$):

T	300	400	500	600
C_p	52.89	65.61	78.07	99.24

Используя интерполяционный полином Лагранжа, необходимо вычислить значение теплоемкости в точке $T = 450$ K.
Применим формулу (8), подставив в нее табличные значения и температуру, при которой нужно определить теплоемкость:

$$f(450) = 52.89 \cdot \frac{(450-400)\cdot(450-500)\cdot(450-600)}{(300-400)\cdot(300-500)\cdot(300-600)} + \\ + 65.61 \cdot \frac{(450-300)\cdot(450-500)\cdot(450-600)}{(400-300)\cdot(400-500)\cdot(400-600)} + \\ + 78.07 \cdot \frac{(450-300)\cdot(450-400)\cdot(450-600)}{(500-300)\cdot(500-400)\cdot(500-600)} + \\ + 99.24 \cdot \frac{(450-300)\cdot(450-400)\cdot(450-500)}{(600-300)\cdot(600-400)\cdot(600-500)} = 71.3119$$

Значение теплоемкости при $T = 450$ K $C_p(450) = 71.3119$ Дж/(моль $\cdot$ К).

Программная реализация


```python
def interpolation(x: list[float], y: list[float], x_: float) -> float:
    product = 1  # переменная для хранения значения произведения
    s = 0        # переменная для хранения значения суммы
    length = len(x)

    for i in range(length):
        for j in range(length):
            if i != j:
                product *= (x_ - x[j]) / (x[i] - x[j])
        s += y[i] * product
        product = 1

    return s


x = [300, 400, 500, 600]
y = [52.89, 65.61, 78.07, 99.24]

cp = interpolation(x, y, 450)
print(cp)  # 71.311875

```

Графическая интерпретация задачи интерполяции

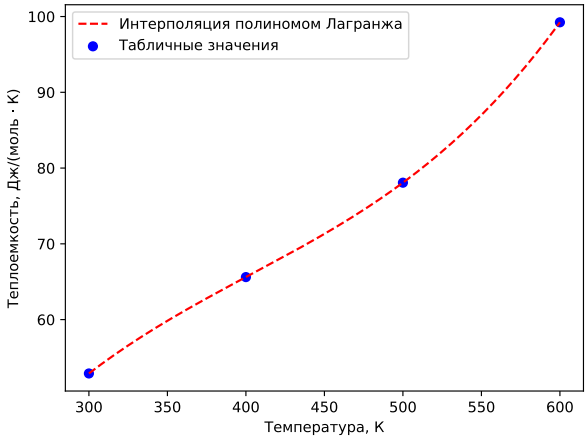

Рисунок 2 – Табличные данные и результаты интерполяции полиномом Лагранжа

TOMSK POLYTECHNIC UNIVERSITY

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Аппроксимация

Аппроксимация

- Отличительной особенностью интерполяции, является тот факт, что она проходит через все узловые точки в исходных данных. Иными словами, рассчитанные по интерполяции данные строго совпадают с табличными значениями: $y_i = f(x_i)$.
- Этот факт объясняется тем, что количество коэффициентов интерполирующей функции (m) совпадает с количеством табличных значений (n).
- Однако на практике часто бывает необходимо выбрать функцию с меньшим количеством коэффициентов ($m < n$) для описания исходных данных, поэтому в этих случаях выбранная функция уже не будет проходить через каждое табличное значение, она может лишь пройти максимально близко к этим значениям (рисунок 3).
- Данный способ описания табличных данных называют **аппроксимацией**, а функцию – **аппроксимирующей**.

Аппроксимация

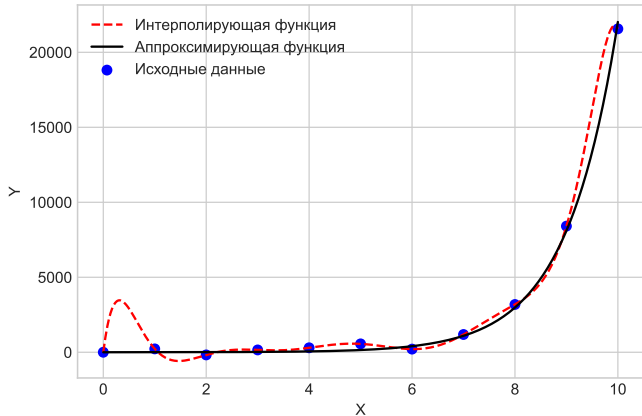

Рисунок 3 – Исходные данные, результаты интерполяции и аппроксимации

Аппроксимация

Есть целый ряд случаев, когда аппроксимация подойдет гораздо лучше, чем интерполяция.

1. Если количество табличных данных очень велико, интерполирующая функция получится крайне громоздкой. Лучше выбрать более простую в реализации функцию с малым числом коэффициентов, пусть и менее точную (хотя это тоже под вопросом).
2. Иногда вид функции заранее известен. Например, константа скорости химический реакции зависит от температуры по уравнению Аррениуса: $k = k_0 \cdot \exp\left(-E_a / \left(R \cdot T\right)\right)$, в котором есть два определяемых параметра: k_0 – предэкспоненциальный множитель и E_a – энергия активации. С большой долей вероятности экспериментальных точек будет больше, чем две, поэтому в данном случае возникает задача аппроксимации.
3. Аппроксимирующая функция сглаживает погрешности, допущенные при эксперименте, в отличие от интерполирующей функции. Обратимся еще раз к рисунку 3. Точками показаны исходные данные, полученные в результате некоторого эксперимента. Нет сомнений в том, что Y просто монотонно возрастает при увеличении X, а разброс данных обусловлен погрешностью, допущенной при проведении эксперимента или при обработке его результатов. Интерполирующая функция будет повторять эту погрешность, проходя через каждую точку, при этом имея множество локальных экстремумов, искажающих характер зависимости $Y(X)$. В это же время аппроксимирующая функция лишена этих недостатков и показывает более адекватное отображение зависимости Y от X.
4. Интерполяция не может быть применена к табличным данным, в которых есть повторяющиеся значения аргумента. Данная ситуация вполне вероятна, особенно в тех случаях, когда один эксперимент проводился несколько раз при одинаковых условиях.

Постановка задачи

Предположим, что имеется ряд измерений величин x и y для неизвестной функциональной зависимости $y = f(x)$.

x	x_1	x_2	x_3	...	x_n
y	y_1	y_2	y_3	...	y_n

В том случае, когда аналитическое выражение функции $f(x)$ неизвестно или сложно, возникает задача аппроксимации, а именно, требуется найти такую эмпирическую функцию

$$\tilde{y} = \tilde{f}(x) \tag{9}$$

для которой значения $\tilde{y}$ при $x = x_i$ как можно меньше отличались бы от экспериментальных (табличных) данных $y_i (i = 1, 2, \ldots, n)$.

В большинстве случаев искомая функция $\tilde{f}(x)$ выбирается из достаточно узкого класса функций: линейных, степенных, показательных и полиномиальных. Следовательно, задача сводится к нахождению оптимальных параметров для каждого предполагаемого класса аппроксимирующей функции.

Постановка задачи

Геометрически задача аппроксимации сводится к построению кривой $\tilde{f}(x)$, которая как можно ближе должна проходить к системе экспериментальных точек (рисунок 4).

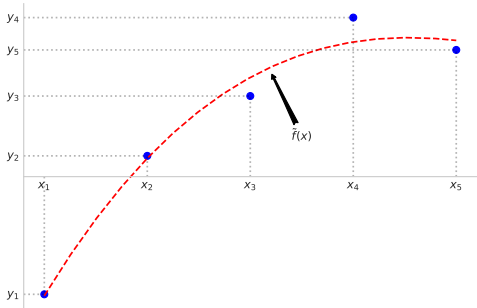

Рисунок 4 – Геометрическая интерпретация задачи аппроксимации

- В тех случаях, когда неизвестен характер зависимости x и y, тип аппроксимирующей функции выбирается произвольно. Предпочтительнее выбирать простые зависимости с хорошей точностью.
- Наиболее распространенными типами аппроксимирующих функций являются линейный, показательный, экспоненциальный и полиномиальный.

Задача определения оптимальных коэффициентов аппроксимирующей функции решается регулярными методами, например, **методом наименьших квадратов**.

Метод наименьших квадратов

Предположим, что в результате эксперимента была получена таблица значений функции $y_i(i = 1, 2, \ldots, n)$. Требуется найти аппроксимирующую функцию неизвестной функциональной зависимости $y(x)$:

$$y(x) = \tilde{f}(x; a_0, a_1, \ldots, a_m) \tag{10}$$

где m – число коэффициентов; $a_1, \ldots, a_m$ – неизвестные коэффициенты.

Используем метод наименьших квадратов для нахождения неизвестных коэффициентов. Идея метода наименьших квадратов заключается в следующем: необходимо определить искомые коэффициенты a_j из выражения (10) таким образом, чтобы эта функция максимально точно описывала экспериментальные данные, т.е. сумма квадратов отклонений экспериментальных значений y_i от соответствующих значений, вычисленных по аппроксимирующей функции (10), была минимальной:

$$F(a_0, a_1, \ldots, a_m) = \sum_{i=1}^{n} \left[y_i - \tilde{f}(x_i; a_0, a_1, \ldots, a_m) \right]^2 \to min \tag{11}$$

где $F(a_0, a_1, \ldots, a_m)$ – функция коэффициентов.

Метод наименьших квадратов

Так как нам требуется найти точку минимума функции F, будем искать точку в которой частные производные по коэффициентам $(a_0, a_1, \ldots, a_m)$ равны нулю. Основываясь на данном положении, получаем так называемую систему для определения коэффициентов $a_j (j = 0, 1, \ldots, m)$:

$$\begin{cases} \dfrac{\partial F}{\partial a_0} = 0 \\ \dfrac{\partial F}{\partial a_1} = 0 \\ \ldots \\ \dfrac{\partial F}{\partial a_m} = 0 \end{cases} \tag{12}$$

Система (12) значительно может быть упрощена, если функция $\tilde{f}(x; a_0, a_1, \ldots, a_m)$ линейна относительно коэффициентов $a_0, a_1, \ldots, a_m$.

Метод наименьших квадратов

Рассмотрим часто встречающийся случай, при котором аппроксимирующая функция представлена полиномом.

$$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_m \cdot x^m$$
$$F(a_0, a_1, \cdot, a_m) = \sum_{i=1}^{n} [y_i - P(x_i)]^2 \to min \quad (13)$$

Используя систему (12), получим математическое условие минимума для уравнения (13):

$$\begin{cases} \dfrac{\partial F}{\partial a_0} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - \left(a_0 - a_1 \cdot x_i + a_2 \cdot x_i^2 + \ldots + a_m \cdot x_i^m\right)\right) \cdot 1 = 0 \\ \dfrac{\partial F}{\partial a_1} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - \left(a_0 - a_1 \cdot x_i + a_2 \cdot x_i^2 + \ldots + a_m \cdot x_i^m\right)\right) \cdot x_i = 0 \\ \ldots \\ \dfrac{\partial F}{\partial a_m} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - \left(a_0 - a_1 \cdot x_i + a_2 \cdot x_i^2 + \ldots + a_m \cdot x_i^m\right)\right) \cdot x_i^m = 0 \end{cases} \quad (14)$$

После решения системы уравнений (14) получим коэффициенты $a_0, a_1, \ldots, a_m$ для полинома (13).

Линейная аппроксимация

На практике часто бывает необходимо построить линейную аппроксимацию экспериментальных данных, т.е. описать закон изменения зависимой переменной линейным уравнением (рисунок 5):

$$P_1(x) = a_0 + a_1 \cdot x \qquad (15)$$

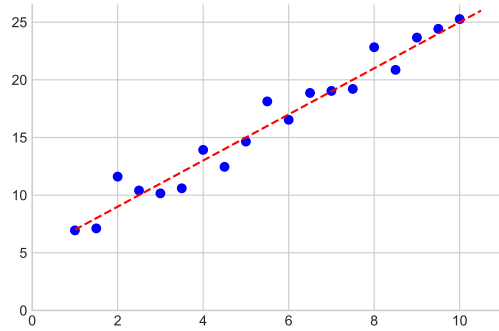

Рисунок 5 – Линейная аппроксимация

Линейная аппроксимация

Рассмотрим вывод формул для расчета коэффициентов a_0 и a_1 линейной аппроксимирующей функции (15) с использованием метода наименьших квадратов.

$$F = \sum_{i=1}^{n} (y_i - a_0 - a_1 \cdot x_i)^2 \to min \tag{16}$$

$$\begin{cases} \dfrac{\partial F}{\partial a_0} = -2 \cdot \sum\limits_{i=1}^{n} (y_i - a_0 - a_1 \cdot x_i) \cdot 1 = 0 \\ \dfrac{\partial F}{\partial a_1} = -2 \cdot \sum\limits_{i=1}^{n} (y_i - a_0 - a_1 \cdot x_i) \cdot x_i = 0 \end{cases}$$

$$\begin{cases} \sum\limits_{i=1}^{n} y_i - a_0 \cdot n - a_1 \cdot \sum\limits_{i=1}^{n} x_i = 0 \\ \sum\limits_{i=1}^{n} (y_i \cdot x_i) - a_0 \cdot \sum\limits_{i=1}^{n} x_i - a_1 \cdot \sum\limits_{i=1}^{n} x^2 = 0 \end{cases}$$

$$\begin{cases} a_0 \cdot n + a_1 \cdot \sum\limits_{i=1}^{n} x_i = \sum\limits_{i=1}^{n} y_i \\ a_0 \cdot \sum\limits_{i=1}^{n} x_i + a_1 \cdot \sum\limits_{i=1}^{n} x^2 = \sum\limits_{i=1}^{n} (x_i \cdot y_i) \end{cases}$$

Линейная аппроксимация

Выразим коэффициенты a_0 и a_1

$$a_0 = \frac{\begin{vmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} (x_i \cdot y_i) & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{\sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} (x_i \cdot y_i)}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2}$$

$$a_1 = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} (x_i \cdot y_i) \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{n \cdot \sum_{i=1}^{n} (x_i \cdot y_i) - \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2} \quad (17)$$

Линейная аппроксимация

Сделаем следующую замену:

$$S_1 = \sum_{i=1}^{n} x_i \qquad S_2 = \sum_{i=1}^{n} y_i$$
$$S_3 = \sum_{i=1}^{n} x_i^2 \qquad S_4 = \sum_{i=1}^{n} x_i \cdot y_i \tag{18}$$

Тогда коэффициенты аппроксимации запишутся следующим образом:

$$a_0 = \frac{S_2 \cdot S_3 - S_1 \cdot S_4}{n \cdot S_3 - S_1^2} \qquad a_1 = \frac{n \cdot S_4 - S_1 \cdot S_2}{n \cdot S_3 - S_1^2} \tag{19}$$

Определитель системы записывается следующим образом:

$$D = \begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix} \neq 0$$

Определение коэффициентов a_0 и a_1 возможно только в том случае, если определитель системы $D \neq 0$, в противном случае система несовместна (не имеет решений) или не определена (имеет бесконечно много решений).

Пример использования линейной аппроксимации

Дана табличная зависимость теплоемкости вещества от температуры.

T, K	300	400	500	600	700	800
C_p, Дж / (моль · K)	6.97	7.01	7.12	7.28	7.45	7.62

Необходимо построить линейную аппроксимирующую функцию и найти значение теплоемкости при температуре $T = 750$ K.
Используя формулы (18), определим значения параметров $S_1,\ S_2,\ S_3,\ S_4$:

$$S_1 = 3300; \qquad S_2 = 43.45; \qquad S_3 = 1990000; \qquad S_4 = 24134.0$$

Далее по формуле (19) определим коэффициенты для линейной функции a_0 и a_1:

$$a_0 = \frac{43.45 \cdot 1990000 - 3300 \cdot 24134.0}{6 \cdot 1990000 - 3300^2} = 6.498380952380953$$
$$a_1 = \frac{6 \cdot 24134.0 - 3300 \cdot 43.45}{6 \cdot 1990000 - 3300^2} = 0.0013514285714285713$$

Подставим полученные коэффициенты и значение температуры, для которой требуется вычислить теплоемкость в уравнение прямой и вычислим искомое значение теплоемкости:

$$C_p(750) = 6.498380952380953 + 0.0013514285714285713 \cdot 750 = 7.511952380952382$$

Программная реализация


```python
def linear_fit(x: list[float], y: list[float]) -> tuple[float, float]:
    n = len(x)
    s1, s2 = sum(x), sum(y)
    s3 = sum(item ** 2 for item in x)
    s4 = sum(item1 * item2 for item1, item2 in zip(x, y))
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return a0, a1


def line(x: float, a0: float, a1: float) -> float:  # уравнение прямой
    return a0 + a1 * x


x = [300, 400, 500, 600, 700, 800]
y = [6.97, 7.01, 7.12, 7.28, 7.45, 7.62]

linear_params = linear_fit(x, y)
cp = line(750, *linear_params)
print(cp)  # 7.511952380952382

```

Программная реализация

С использованием массивов NumPy:

```python
import numpy as np

def linear_fit(x: np.ndarray, y: np.ndarray) -> tuple[float, float]:
    n, = x.shape  # распаковка кортежа из одного элемента
    s1, s2 = x.sum(), y.sum()
    s3 = (x ** 2).sum()
    s4 = (x * y).sum()
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return a0, a1

def line(x: float, a0: float, a1: float) -> float:  # уравнение прямой
    return a0 + a1 * x

x = np.array([300, 400, 500, 600, 700, 800])
y = np.array([6.97, 7.01, 7.12, 7.28, 7.45, 7.62])
linear_params = linear_fit(x, y)
cp = line(750, *linear_params)
print(cp)  # 7.511952380952382
```

Графическое отображение

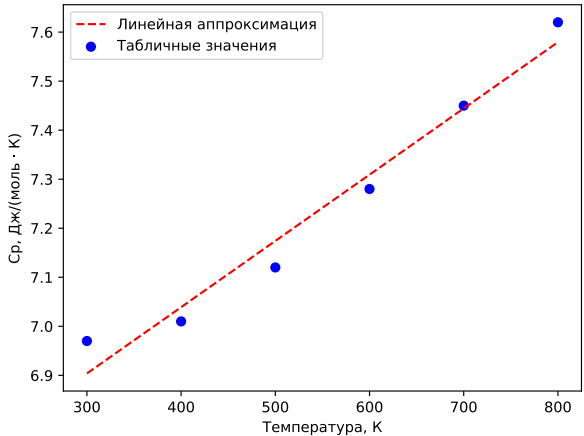

Рисунок 6 – Графическая оценка качества линейной аппроксимации

Аппроксимация экспоненциальной функцией

Экспоненциальная функция представлена зависимостью следующего вида:

$$y = a \cdot e^{b \cdot x} \tag{20}$$

где a, b – неизвестные коэффициенты.
Прологарифмируем уравнение (20) для приведения его к линейному виду:

$$\ln(y) = \ln(a) + b \cdot x$$

и введем следующие обозначения: $Y = \ln(y)$; $A_0 = \ln(a)$; $A_1 = b$, получим следующее линейное уравнение:

$$Y = A_0 + A_1 \cdot x \tag{21}$$

Воспользуемся методом наименьших квадратов:

$$F = \sum_{i=1}^{n} [Y_i - (A_0 + A_1 \cdot x_i)]^2 \to min$$

Так как исходная функция приведена к линейному виду, последовательность вычисления коэффициентов A_0 и A_1 выполняется по формулам (19). Далее сделаем обратную замену:

$$a = e^{A_0}; \qquad b = A_1; \qquad y_i = e^{Y_i}.$$

Пример

Решим пример, приведенный на слайде 28 с использованием экспоненциальной аппроксимации.

```python
import numpy as np

def exp_fit(x: list[float], y: list[float]) -> tuple[float, float]:
    n = len(x)
    s1 = sum(x)
    s2 = sum(np.log(item) for item in y)
    s3 = sum(item ** 2 for item in x)
    s4 = sum(item1 * np.log(item2) for item1, item2 in zip(x, y))
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return np.exp(a0), a1

def exponent(x: float, a0: float, a1: float) -> float:
    return a0 * np.exp(a1 * x)

x = [300, 400, 500, 600, 700, 800]
y = [6.97, 7.01, 7.12, 7.28, 7.45, 7.62]
exp_params = exp_fit(x, y)
cp = exponent(750, *exp_params)
print(cp)  # 7.51206780537444
```

Пример

С помощью массивов NumPy:

```python
import numpy as np

def exp_fit(x: np.ndarray, y: np.ndarray) -> tuple[float, float]:
    n, = x.shape
    s1 = x.sum()
    s2 = np.log(y).sum()
    s3 = (x ** 2).sum()
    s4 = (x * np.log(y)).sum()
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return np.exp(a0), a1

def exponent(x: float, a0: float, a1: float) -> float:
    return a0 * np.exp(a1 * x)

x = np.array([300, 400, 500, 600, 700, 800])
y = np.array([6.97, 7.01, 7.12, 7.28, 7.45, 7.62])
exp_params = exp_fit(x, y)
cp = exponent(750, *exp_params)
print(cp)  # 7.51206780537444
```

Графическое отображение

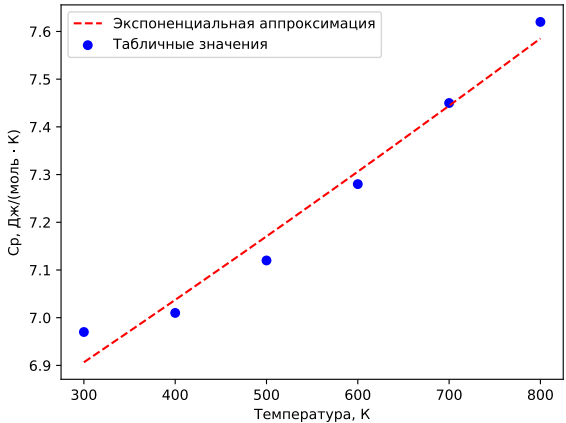

Рисунок 7 – Графическая оценка качества экспоненциальной аппроксимации

Аппроксимация степенной функцией

Степенная функция записывается следующим образом:

$$y = a \cdot x^b \tag{22}$$

Приведем данное уравнение к линейному виду, прологарифмировав его:

$$\ln(y_i) = \ln(a) + b \cdot \ln(x_i) \tag{23}$$

Сделаем следующую замену: $Y = \ln(y)$; $A_0 = \ln(a_0)$; $A_1 = b$; $X = \ln(x)$.
Применим метод наименьших квадратов:

$$F = \sum_{i=1}^{n} [Y_i - (A_0 + A_1 \cdot X_i)]^2 \to min$$

Поскольку мы привели исходную степенную функцию к линейному виду, последовательность вычисления коэффициентов A_0 и A_1 полностью совпадает со случаем линейной аппроксимации, т.е. их вычисление выполняется по формулам (19).
После определения коэффициентов необходимо сделать обратную замену:

$$a_0 = e^{A_0}; \qquad b = A_1; \qquad y_i = e^{Y_i}; \qquad x_i = e^{X_i}.$$

Пример

Решим пример, приведенный на слайде 28 с использованием степенной аппроксимации.

```python
import numpy as np

def pow_fit(x: list[float], y: list[float]) -> tuple[float, float]:
    n = len(x)
    s1 = sum(np.log(item) for item in x)
    s2 = sum(np.log(item) for item in y)
    s3 = sum(np.log(item) ** 2 for item in x)
    s4 = sum(np.log(item1) * np.log(item2) for item1, item2 in zip(x, y))
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return np.exp(a0), a1

def power(x: float, a0: float, a1: float) -> float:
    return a0 * x ** a1

x = [300, 400, 500, 600, 700, 800]
y = [6.97, 7.01, 7.12, 7.28, 7.45, 7.62]
pow_params = pow_fit(x, y)
cp = power(750, *pow_params)
print(cp)  # 7.483573398167825
```

Пример

С помощью массивов NumPy:

```python
import numpy as np

def pow_fit(x: list[float], y: list[float]) -> tuple[float, float]:
    n, = x.shape
    s1 = np.log(x).sum()
    s2 = np.log(y).sum()
    s3 = (np.log(x) ** 2).sum()
    s4 = (np.log(x) * np.log(y)).sum()
    a0 = (s2 * s3 - s1 * s4) / (n * s3 - s1 ** 2)
    a1 = (n * s4 - s1 * s2) / (n * s3 - s1 ** 2)
    return np.exp(a0), a1

def power(x: float, a0: float, a1: float) -> float:
    return a0 * x ** a1

x = np.array([300, 400, 500, 600, 700, 800])
y = np.array([6.97, 7.01, 7.12, 7.28, 7.45, 7.62])
pow_params = pow_fit(x, y)
cp = power(750, *pow_params)
print(cp)  # 7.483573398167825
```

Графическое отображение

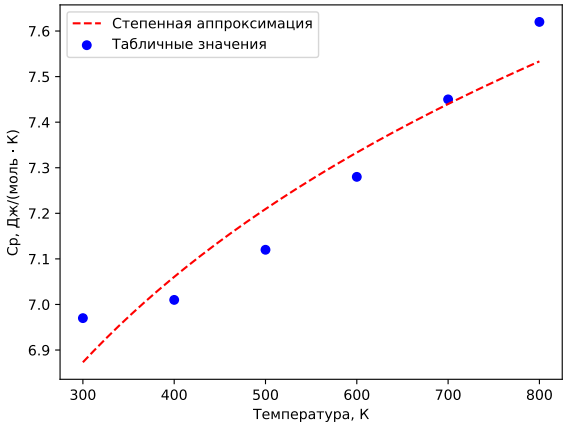

Рисунок 8 – Графическая оценка качества степенной аппроксимации

Параболическая аппроксимация

В тех случаях, когда линейная аппроксимация не описывает экспериментальные данные с требуемой точностью можно использовать аппроксимацию полиномом второй степени (квадратичную или параболическую) либо полиномом более высокой степени

Полином второй степени имеет следующий вид:

$$P_2(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 \tag{24}$$

По аналогии с линейной аппроксимацией, коэффициенты a_j определяются при помощи метода наименьших квадратов:

$$F = \sum_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2\right)^2 \to min \tag{25}$$

Параболическая аппроксимация

Приравняем к нулю частные производные:

$$\begin{cases} \dfrac{\partial F}{\partial a_0} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2\right) \cdot 1 = 0 \\ \dfrac{\partial F}{\partial a_1} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2\right) \cdot x_i = 0 \\ \dfrac{\partial F}{\partial a_2} = -2 \cdot \sum\limits_{i=1}^{n} \left(y_i - a_0 - a_1 \cdot x_i - a_2 \cdot x_i^2\right) \cdot x_i^2 = 0 \end{cases} \tag{26}$$

После преобразований получим систему линейных уравнений с тремя неизвестными a_0, a_1 и a_2:

$$\begin{cases} a_0 \cdot n + a_1 \cdot \sum\limits_{i=1}^{n} x_i + a_2 \cdot \sum\limits_{i=1}^{n} x_i^2 = \sum\limits_{i=1}^{n} y_i \\ a_0 \cdot \sum\limits_{i=1}^{n} x_i + a_1 \cdot \sum\limits_{i=1}^{n} x_i^2 + a_2 \cdot \sum\limits_{i=1}^{n} x_i^3 = \sum\limits_{i=1}^{n} (x_i \cdot y_i) \\ a_0 \cdot \sum\limits_{i=1}^{n} x_i^2 + a_1 \cdot \sum\limits_{i=1}^{n} x_i^3 + a_2 \cdot \sum\limits_{i=1}^{n} x_i^4 = \sum\limits_{i=1}^{n} \left(x_i^2 \cdot y_i\right) \end{cases} \tag{27}$$

Параболическая аппроксимация

Введем следующие обозначения:

$$S_1 = \sum_{i=1}^{n} x_i; \qquad S_2 = \sum_{i=1}^{n} x_i^2; \qquad S_3 = \sum_{i=1}^{n} x_i^3; \qquad S_4 = \sum_{i=1}^{n} x_i^4;$$
$$S_5 = \sum_{i=1}^{n} y_i; \qquad S_6 = \sum_{i=1}^{n} (x_i \cdot y_i); \qquad S_7 = \sum_{i=1}^{n} (x_i^2 \cdot y_i) \tag{28}$$

После введения обозначений система (27) примет следующий вид:

$$\begin{cases} a_0 \cdot n + a_1 \cdot S_1 + a_2 \cdot S_2 = S_5 \\ a_0 \cdot S_1 + a_1 \cdot S_2 + a_2 \cdot S_3 = S_6 \\ a_0 \cdot S_2 + a_1 \cdot S_3 + a_2 \cdot S_4 = S_7 \end{cases} \tag{29}$$

Параболическая аппроксимация

Найдем неизвестные коэффициенты a_0, a_1 и a_2:

$$a_0 = \frac{\begin{vmatrix} S_5 & S_1 & S_2 \\ S_6 & S_2 & S_3 \\ S_7 & S_3 & S_4 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}, \quad a_1 = \frac{\begin{vmatrix} n & S_5 & S_2 \\ S_1 & S_6 & S_3 \\ S_2 & S_7 & S_4 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}, \quad a_2 = \frac{\begin{vmatrix} n & S_1 & S_5 \\ S_1 & S_2 & S_6 \\ S_2 & S_3 & S_7 \end{vmatrix}}{\begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix}}. \tag{30}$$

Необходимое условие: определитель не должен быть равен 0:

$$D = \begin{vmatrix} n & S_1 & S_2 \\ S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \end{vmatrix} \neq 0.$$

Пример

Решим пример, приведенный на слайде 28 с использованием квадратичной аппроксимации. Воспользуемся формулами (28) для определения параметров $S_1,\ S_2,\ S_3,\ S_4,\ S_5,\ S_6,\ S_7$:

$$S_1 = 3300; \qquad S_2 = 1990000; \qquad S_3 = 1287000000; \qquad S_4 = -673328384;$$
$$S_5 = 43.45; \qquad S_6 = 24134.0; \qquad S_7 = 14677000.0$$

По формулам (30) определим коэффициенты a_0, a_1 и a_2:

$$a_0 = 6.496522; \qquad a_1 = 0.001359; \qquad a_2 = -6.800850 \times 10^{-9}$$

Таким образом, аппроксимирующее уравнение примет следующий вид:

$$y = 6.496522 + 0.001359 \cdot x - 6.800850 \times 10^{-9} \cdot x^2$$

Проведем оценку качества полученной аппроксимирующей функции:

T, К	300	400	500	600	700	800
C_p, Дж/(моль · К)	6.97	7.01	7.12	7.28	7.45	7.62
$C_p^{\text{расчет}}$, Дж/(моль · К)	6.90	7.04	7.17	7.31	7.44	7.58
$C_p - C_p^{\text{расчет}}$, Дж/(моль · К)	0.07	−0.03	−0.05	−0.03	0.01	0.04

Программная реализация


```python
import numpy as np

def poly2_fit(x: np.ndarray, y: np.ndarray) -> tuple[float, float, float]:
    n, = x.shape
    # Определение параметров S1-S7
    s1, s2, s3, s4 = (
        x.sum(), (x ** 2).sum(), (x ** 3).sum(), (x ** 4).sum()
    )
    s5, s6, s7 = (
        y.sum(), (x * y).sum(), (x ** 2 * y).sum()
    )
    # Определение коэффициентов a0, a1, a2 с помощью функции np.linalg.det
    a0 = (np.linalg.det(np.array([[s5, s1, s2], [s6, s2, s3], [s7, s3, s4]]))
        / np.linalg.det(np.array([[n, s1, s2], [s1, s2, s3], [s2, s3, s4]])))
    a1 = (np.linalg.det(np.array([[n, s5, s2], [s1, s6, s3], [s2, s7, s4]]))
        / np.linalg.det(np.array([[n, s1, s2], [s1, s2, s3], [s2, s3, s4]])))
    a2 = (np.linalg.det(np.array([[n, s1, s5], [s1, s2, s6], [s2, s3, s7]]))
        / np.linalg.det(np.array([[n, s1, s2], [s1, s2, s3], [s2, s3, s4]])))
    return a0, a1, a2
```

Программная реализация


```
20
21
22  def poly2(params: tuple[float, float, float], x: float) -> float:
23      a0, a1, a2 = params
24      return a0 + a1 * x + a2 * x ** 2
25
26
27  x = np.array([300, 400, 500, 600, 700, 800])
28  y = np.array([6.97, 7.01, 7.12, 7.28, 7.45, 7.62])
29  poly2_params = poly2_fit(x, y)
30  y_ = poly2(poly2_params, x)
31  print(y_)
32
```

```
[6.90358283 7.03899772 7.17427659 7.30941945 7.44442629 7.57929711]
```

Графическое отображение

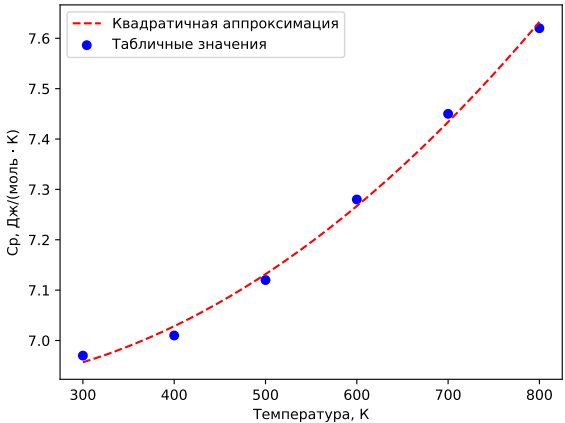

Рисунок 9 – Графическая оценка качества квадратичной аппроксимации

Класс `Polynomial` библиотеки NumPy

- Класс `Polynomial` предоставляет удобный естественный интерфейс во внутренней функциональности пакета `polynomial` библиотеки NumPy. Чтобы импортировать его напрямую, используйте инструкцию:

```
1 from numpy.polynomial import Polynomial
2
```

- Или если вся библиотека NumPy уже импортирована как `np`, то вместо того, чтобы часто обращаться к этому классу как `np.polynomial.Polynomial`, удобнее определить переменную:

```
1 import numpy as np
2 Polynomial = np.polynomial.Polynomial
3
```

- Для определения объекта полинома необходимо передать в конструктор `Polynomial` последовательность коэффициентов для постепенно возрастающих степеней x, начиная с c_0. Например, для представления многочлена

$$P(x) = 6 - 5x + x^2$$

определяется объект

```
4 p = Polynomial([6, -5, 1])
5
```

Класс `Polynomial` библиотеки NumPy

- Для вычисления полинома при заданном значении x необходимо «вызвать» полином, как показано ниже:

```
6  y1 = p(4)
7  x = np.linspace(-5, 5, 11)
8  y2 = p(x)
9  print(y1)
10 print(y2)
11
```

```
2.0
[ 56. 42. 30. 20. 12. 6. 2. 0. 0. 2. 6.]
```

- Корни полинома возвращает метод `roots()`:

```
12 roots = p.roots()
13 print(roots)
14
```

```
[2., 3.]
```

Метод `Polynomial.fit()`

- Метод класса `Polynomial.fit()` возвращает полином, подогнанный методом наименьших квадратов к данным из выборки значений y.
- Для метода `fit()` требуется передача массивов x и y и значения deg – степени полинома.
- Метод возвращает полином, который минимизирует сумму квадратов ошибок:

$$E = \sum_{i=1}^{n} [y_i - p(x_i)]^2$$

Рассмотрим функцию $y = e^{-\sin x}$ на отрезке $[1; 12]$.

```
1 import numpy as np
2 Polynomial = np.polynomial.Polynomial
3
4
5 x = np.linspace(1, 12, 10)
6 y = np.exp(-np.sin(x))
7 p3 = Polynomial.fit(x, y, 3)  # Создает кубический полином
8 print(p3.coef)
```

```
[ 1.29952606 -0.96210884 -0.04580373  2.00818378]
```

Метод `Polynomial.fit()`

- Следует помнить о том, что для полинома высокой степени характерно большое количество локальных экстремумов, что создаст проблемы при аппроксимации исходных данных.

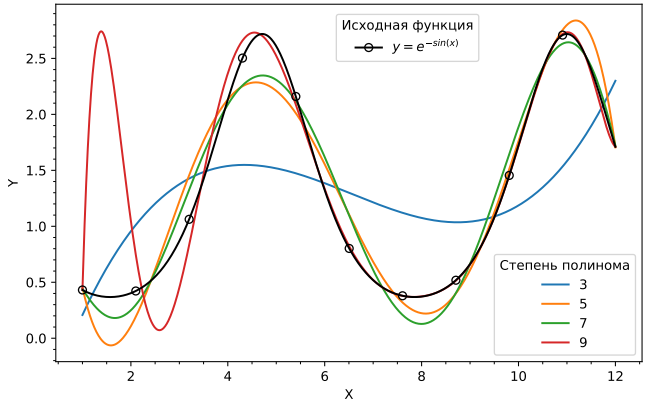

Рисунок 10 – Аппроксимация функции $y = e^{-\sin x}$ полиномиальными функциями

TOMSK POLYTECHNIC UNIVERSITY
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Вячеслав Алексеевич Чузлов
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
chuva@tpu.ru
+7-962-782-66-15

Благодарю за внимание!