ГЛАВА 1

Обзор начальной математики и математического анализа

В начале первой главы мы разберемся, что такое числа и как устроены переменные и функции в декартовой системе координат. Затем мы рассмотрим возведение в степень и логарифмы, а после этого изучим две основные конструкции математического анализа — производные и интегралы.

Прежде чем углубиться в прикладные области математики — такие, как теория вероятностей, линейная алгебра, математическая статистика и машинное обучение, — имеет смысл рассмотреть некоторые основные понятия базовой математики и математического анализа. Только прошу, не бросайте эту книгу и не разбегайтесь в ужасе! Я расскажу, как вычислять производные и интегралы для функции таким способом, которому вас вряд ли учили в университете. Мы будем делать это не на бумаге, а с помощью языка программирования Python. Даже если вы не знакомы с производными и интегралами, не переживайте.

Я постараюсь изложить эти темы максимально сжато, уделяя внимание только тому, что пригодится в последующих главах и что относится к теме математики для data science.

Это не полноценный курс математики!

Эта книга ни в коем случае не претендует на то, чтобы служить исчерпывающим обзором математики для средних и высших учебных заведений. Если вам нужна именно такая книга, то обратите внимание на «*No Bullshit Guide to Math and Physics*» Ивана Савова (Ivan Savov). Первые несколько глав содержат лучший краткий курс по математике из всех, что я когда-либо видел. Книга «*Mathematics 1001*» Ричарда Элвиса (Richard Elwes) тоже отличается замечательным материалом, причем в виде небольших заметок.

Теория чисел

Что такое числа? Я обещаю не слишком философствовать в этой книге, но не являются ли числа искусственной конструкцией, которую мы сами же и придумали? Почему мы пользуемся цифрами от 0 до 9, а не больше? Почему у нас есть обыкновенные и десятичные дроби, а не только целые числа? Область математики, которая занимается числами и отвечает на вопрос, почему мы определили их так, а не иначе, известна как теория чисел, или высшая арифметика.

Теория чисел уходит корнями в глубокую древность, когда математики открывали различные числовые системы, и объясняет, почему сейчас мы используем числа именно в таком виде. Ниже представлены несколько числовых систем, которые наверняка вам знакомы.

Натуральные числа

Это числа 1, 2, 3, 4, 5 и т. д.[1] Сюда входят только положительные числа, и это самая ранняя известная нам числовая система. Даже пещерные люди выцарапывали на костях и стенах пещер насечки, которые изображали натуральные числа, — так наши предки вели свои первобытные подсчеты.

Целые неотрицательные числа

В дополнение к натуральным числам позднее появилось понятие нуля и соответствующий символ «0». Целые неотрицательные числа — это 0, 1, 2, 3 и т. д. Из Древнего Вавилона пришла идея обозначать особым символом пустые «разряды» в многозначных числах, таких как 10, 1 000 или 1 090. Нули в них указывают на то, что в соответствующем разряде нет значения.

Целые числа

К целым числам относятся натуральные числа, противоположные им отрицательные числа, а также 0. Для нас это понятие — само собой разумеющееся, но древние математики с глубоким недоверием относились к идее отрицательных чисел. Однако если из 3 вычесть 5, то получится −2. Это особенно удобно, когда речь идет о финансах, где мы измеряем прибыль и убытки. В 628 году нашей эры индийский математик Брахмагупта на примере решения квадратного уравнения показал, почему отрицательные числа необходимы для развития арифметики, и только после этого они стали общепринятыми.

---

[1] Иногда к натуральным числам относят и ноль: например, стандарты ISO 80000-2 и ГОСТ Р 54521–2011 определяют два множества натуральных чисел: $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ и $\mathbb{N}^* = \{0, 1, 2, 3, \ldots\}$. В этой книге принят весьма распространенный и удобный на практике подход, согласно которому натуральный ряд начинается с 1. — *Примеч. науч. ред.*

Рациональные числа

Любое число, которое можно представить в виде обыкновенной дроби, например $\frac{2}{3}$, относится к рациональным. Сюда же входят все конечные десятичные дроби и целые числа, потому что их тоже можно выразить в виде обыкновенной дроби, например $\frac{687}{100} = 6,87$ и $\frac{2}{1} = 2$ соответственно. *Рациональными* они называются потому, что одно из значений латинского слова *ratiō* — «отношение, пропорция». Рациональные числа быстро стали необходимы, потому что время, количество вещества и другие величины не всегда можно измерить в дискретных единицах. Например, молоко не обязательно продается в литровой упаковке; иногда его объем приходится измерять в долях литра. Или, скажем, если я бежал 12 минут, я не смогу выразить пройденное расстояние в целых километрах; оно составит $\frac{7}{5}$ километра.

Иррациональные числа

Иррациональные числа нельзя выразить в виде обыкновенной дроби. К ним относятся знаменитое число ϖ, квадратные корни из некоторых чисел (например $\sqrt{2}$), и число Эйлера[1] e, о котором мы узнаем позже. В десятичной записи у этих чисел бесконечное количество знаков после запятой, например 3,14159265358979323846...

История иррациональных чисел довольно интересна. Древнегреческий математик Пифагор считал, что все числа рациональны. Он верил в это так горячо, что создал религию, которая предписывала поклоняться числу 10. «Благослови нас, божественное число, ты, породившее богов и людей!» — молился он и его последователи (почему число 10 было таким особенным, я не знаю). Существует легенда, что один из пифагорейцев по имени Гиппас доказал, что не все числа рациональны, просто продемонстрировав квадратный корень из 2. Это сильно пошатнуло систему верований Пифагора, и в наказание он утопил Гиппаса в море.

Вещественные числа

К вещественным (или действительным) числам относятся все рациональные и все иррациональные числа. На практике в data science любые десятичные дроби, с которыми вы работаете, можно рассматривать как вещественные числа.

Комплексные и мнимые числа

С такими числами можно столкнуться, если извлекать квадратный корень из отрицательного числа. Хотя мнимые и комплексные числа играют свою роль в некоторых типах задач, мы в основном будем держаться от них подальше.

---

[1] В отечественной практике не принято называть *e* числом Эйлера. — *Примеч. ред.*

В data science практически всегда используются натуральные, целые и вещественные числа. Мнимые числа могут встретиться в более сложных случаях — например, при разложении матриц, которых мы коснемся в главе 4.

Комплексные и мнимые числа

Если вы хотите подробнее узнать про мнимые числа, на YouTube есть отличный набор видеороликов «*Imaginary Numbers are Real*» («Мнимые числа реальны», https://oreil.ly/bvyIq).

Порядок выполнения арифметических операций

Надеемся, что вы знакомы с *порядком выполнения* арифметических операций: это порядок, в котором вычисляется каждая часть математического выражения. Напомню, что сначала вычисляется то, что заключено в круглые скобки, затем следует возведение в степень, затем умножение, деление, сложение и вычитание. Операции с одинаковым приоритетом (такие, как несколько сложений подряд) выполняются слева направо.

Например, вычислим значение такого выражения:

$$2\times\frac{(3+2)^2}{5}-4$$

В первую очередь вычислим выражение внутри скобок $(3+2)$, которое дает 5:

$$2\times\frac{(\mathbf{5})^2}{5}-4$$

Затем возведем в квадрат число 5, которое мы только что получили:

$$2\times\frac{\mathbf{25}}{5}-4$$

Далее следуют умножение и деление. Их порядок можно менять местами, потому что деление — это то же самое, что умножение, но с использованием дробей.

Умножая 2 на $\frac{25}{5}$, получаем $\frac{50}{5}$:

$$\frac{50}{5}-4$$

Затем разделим 50 на 5, в результате чего получим 10:

$$\mathbf{10}-4$$

И, наконец, выполняем сложение и вычитание. Естественно, 10 – 4 даст нам 6:

$$10-4=6$$

Конечно, если бы мы сформулировали это выражение на Python, то получили бы значение `6.0`, как показано в примере 1.1.

Пример 1.1. Вычисление выражения на Python

```
my_value = 2 * (3 + 2)**2 / 5 - 4
print(my_value)       # выводит 6.0
```

Следующее замечание может показаться элементарным, но все же об этом важно помнить. Хорошим тоном считается использовать в коде круглые скобки в сложных выражениях, чтобы показать порядок вычисления, даже если формально скобки не обязательны.

В примере 1.2 я сгруппировал дробную часть выражения в круглых скобках, что помогает отделить ее от остального выражения.

Пример 1.2. Использование в Python круглых скобок для наглядности

```
my_value = 2 * ((3 + 2)**2 / 5) - 4
print(my_value)       # выводит 6.0
```

Хотя оба примера синтаксически правильны, второй легче воспринимать человеческим глазом. Когда вы или кто-то другой вносит изменения в код, легко ориентироваться по круглым скобкам, чтобы не нарушить порядок операций. Это обеспечивает дополнительную защиту от ошибок при изменении кода.

Переменные

Если вы программировали на Python или на другом языке, то представляете себе, что такое переменная. В математике *переменная* — это именованная «область» для размещения неопределенного или неизвестного числа.

Например, можно взять переменную x, которая обозначает любое вещественное число, и умножить ее, не уточняя, чему конкретно равно ее значение. В примере 1.3 мы принимаем от пользователя переменную x и умножаем ее на 3.

Пример 1.3. Переменная в Python, над которой производится операция умножения

```
x = int(input("Введите число:\n"))
product = 3 * x
print(product)
```

Для некоторых типов переменных существуют стандартные имена. Ничего страшного, если вы видите их первый раз в жизни! Но некоторые узна́ют греческие

буквы θ (тета), которая обозначает углы, и β (бета), которая обозначает параметр линейной регрессии. Использовать греческие символы в именах переменных в Python довольно неудобно, поэтому мы назовем эти переменные `theta` и `beta`, как показано в примере 1.4[1].

Пример 1.4. Греческие имена переменных в Python

```
beta = 1.75
theta = 30.0
```

Заметим также, что в математике имена переменных могут сопровождаться индексами, чтобы одно и то же имя можно было использовать для нескольких экземпляров переменной. Для практических целей просто считайте, что эти имена принадлежат отдельным переменным. Если вы встречаете переменные x_1, x_2 и x_3, рассматривайте их как три разные переменные, как показано в примере 1.5.

Пример 1.5. Переменные с нижним индексом в Python

```
x1 = 3   # или x_1 = 3
x2 = 10  # или x_2 = 10
x3 = 44  # или x_3 = 44
```

Функции

Функции — это выражения, которые задают соответствие между двумя или более переменными. Более конкретно, функция берет *входные переменные* (которые также называются *независимыми переменными* или *аргументами функции*), подставляет их в выражение, и в результате получается *выходная переменная* (которая также называется *зависимой переменной* или *значением функции*).

Рассмотрим пример простой линейной функции:

$$y = 2x + 1$$

Для любого заданного значения x мы вычисляем выражение с этим x, чтобы найти y. Если $x = 1$, то $y = 3$. Если $x = 2$, то $y = 5$. Если $x = 3$, то $y = 7$ и так далее, как показано в табл. 1.1.

[1] Современный Python формально позволяет использовать в именах переменных более 3 000 символов Unicode; например, θ или мама — допустимые имена. Однако официальная документация по Python рекомендует обходиться только «обычными» символами, которые можно ввести с клавиатуры в латинской раскладке. В большинстве случаев это лучшее решение. — *Примеч. науч. ред.*

Таблица 1.1. Различные значения функции $y = 2x + 1$

x	$2x+1$	y
0	$(2\times0)+1$	1
1	$(2\times1)+1$	3
2	$(2\times2)+1$	5
3	$(2\times3)+1$	7

Функции полезны тем, что они моделируют предсказуемую зависимость между переменными: например, сколько возгораний *y* можно ожидать при температуре *x*. Линейные функции пригодятся в главе 5, где мы будем заниматься линейной регрессией.

Зависимую переменную *y* иногда записывают в другой форме, явно обозначая ее как функцию от *x*, например $f(x)$. Таким образом, вместо того чтобы указывать функцию как $y = 2x + 1$, ее можно выразить так:

$$f(x) = 2x + 1$$

В примере 1.6 показано, как можно объявить математическую функцию в Python и запустить ее.

Пример 1.6. Объявление линейной функции в Python

```
def f(x):
    return 2 * x + 1

x_values = [0, 1, 2, 3]

for x in x_values:
    y = f(x)
    print(y)
```

У функций от вещественных чисел есть порой незаметная, но важная особенность: они часто имеют бесконечное число значений x и соответствующих значений y. Подумайте: сколько значений x можно подставить в функцию $f(x) = 2x + 1$? Почему бы вместо 0; 1; 2; 3 не использовать 0; 0,5; 1; 1,5; 2; 2,5; 3 — как показано в табл. 1.2?

Таблица 1.2. Различные значения функции $y = 2x + 1$

x	$2x+1$	y
0	$(2\times0)+1$	1
0,5	$(2\times0,5)+1$	2
1,0	$(2\times1)+1$	3
1,5	$(2\times1,5)+1$	4
2,0	$(2\times2)+1$	5
2,5	$(2\times2,5)+1$	6
3,0	$(2\times3)+1$	7

Или почему бы не перебирать x с шагом 0,25? Или $\frac{1}{10}$? Эти шаги можно сделать бесконечно малыми, тем самым показывая, что $y = 2x + 1$ — это *непрерывная функция*, где для каждого возможного значения x существует значение y. Это позволяет представить функцию в виде линии, как показано на рис. 1.1.

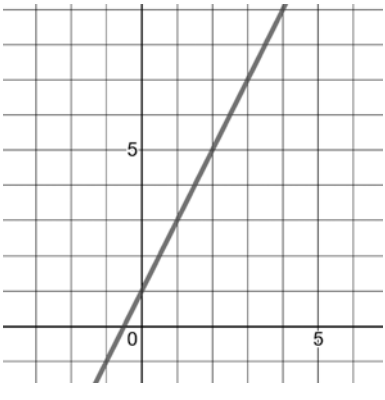

Рис. 1.1. График функции $y = 2x + 1$

Двумерная плоскость с двумя числовыми осями (по одной для каждой переменной), на которой мы строим график, называется *декартовой* (или *прямоугольной*) *системой координат* или *координатной плоскостью*. Мы отслеживаем заданное значение x, затем находим соответствующее значение y и строим пересечения всех таких значений в виде линии. Обратите внимание, что вещественные числа (или десятичные дроби, если вам с ними привычнее) устроены так, что существует

бесконечное количество значений x. Именно поэтому, когда мы строим график функции $f(x)$, получается непрерывная линия без разрывов. На этой линии, как и на любом ее отрезке, находится бесконечное количество точек.

Чтобы строить графики с помощью Python, существует множество библиотек — от Plotly до matplotlib. Во многих задачах на протяжении этой книги мы будем использовать SymPy, и первая задача — построить график функции. SymPy использует matplotlib, поэтому убедитесь, что у вас установлен этот пакет. (В противном случае программа выведет в консоль уродливую диаграмму из текстовых символов.) После этого просто объявите переменную x с помощью функции `symbols()` из SymPy, объявите функцию, а затем постройте график, как показано в примере 1.7 и на рис. 1.2.

Пример 1.7. Построение графика линейной функции с помощью SymPy

```
from sympy import *

x = symbols('x')
f = 2*x + 1
plot(f)
```

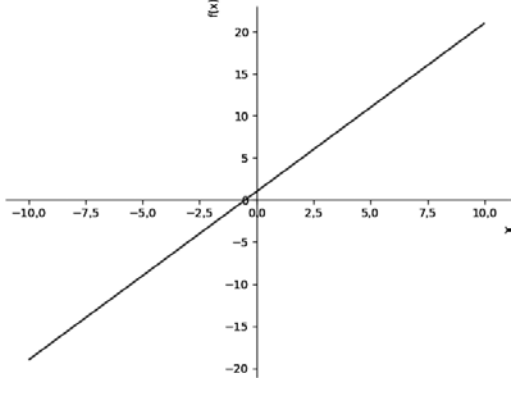

Рис. 1.2. Построение графика линейной функции с помощью SymPy

Пример 1.8 и рис. 1.3 демонстрируют другую функцию: $f(x) = x^2 + 1$.

Пример 1.8. Построение графика квадратичной функции

```
from sympy import *

x = symbols('x')
f = x**2 + 1
plot(f)
```