

Preview of Chapter 02 PROB

from the

NO BULLSHIT GUIDE TO STATISTICS

Ivan Savov

November 8, 2022

Contents

2	Probability	1
2.1	Discrete random variables	7
2.1.1	Definitions	7
2.1.2	Cumulative distribution functions	15
2.1.3	Expected value calculations	18
2.1.4	Computer models for random variables	24
2.1.5	Hard disks example	25
2.1.6	Discussion	31
2.2	Multiple random variables	37
2.2.1	Definitions	37
2.2.2	Joint probability distributions	39
2.2.3	Conditional probability distributions	44
2.2.4	Probability formulas and rules	50
2.2.5	Independent random variables	56
2.2.6	Discussion	59
2.3	Inventory of discrete distributions	62
2.3.1	Math prerequisites	63
2.3.2	Review of definitions and formulas	73
2.3.3	Review of computer models	74
2.3.4	Discrete distributions reference	76
2.3.5	Modelling real-world data	96
2.3.6	Discussion	97
2.4	Calculus prerequisites	102
2.4.1	Definitions	102
2.4.2	Sets and intervals	103
2.4.3	Functions	106
2.4.4	Integrals as area calculations	110
2.4.5	Integrals as functions	113
2.4.6	Computing integrals numerically using SciPy	115
2.4.7	Computing integrals symbolically with SymPy	118
2.4.8	Other calculus topics	121
2.5	Continuous random variables	129

2.5.1	Definitions	129
2.5.2	Cumulative distribution function	132
2.5.3	Calculating expectations	135
2.5.4	Computer models for random variables	141
2.5.5	Kombucha volume example	142
2.5.6	Multiple continuous random variables	148
2.5.7	Discussion	159
2.6	Inventory of continuous distributions	163
2.6.1	Math prerequisites	163
2.6.2	Continuous distributions reference	165
2.6.3	Modelling real-world data	193
2.6.4	Discussion	194
2.6.5	Exercises	196
2.7	Random variable generation	197
2.7.1	Definitions	197
2.7.2	Why simulate?	197
2.7.3	Random variable generation using a computer	199
2.7.4	Empirical distribution of a data sample	204
2.7.5	Measuring data–model fit	206
2.7.6	Bootstrap sample generation	215
2.7.7	Discussion	219
2.8	Probability models for random samples	222
2.8.1	Definitions	222
2.8.2	Sample statistics	223
2.8.3	Sampling distributions of statistics	226
2.8.4	Central limit theorem	235
2.8.5	Discussion	239
2.9	Probability problems	241
2.9.1	Simple probability problems	241
2.9.2	Discrete distributions problems	242
2.9.3	Continuous distributions problems	243
A	Answers and solutions	244
C	Python tutorial	245

Chapter 2

Probability

Probability theory is a language for describing uncertainty, variability, and randomness. Understanding the machinery of probability theory is essential for doing statistics, because probability models allow us to describe the variability that exists in populations and samples. In order to apply statistical analysis procedures correctly, you need to understand the basics of probability theory, and learn about the various probability models that are used as building blocks in statistical analyses. This is what this chapter is all about.

What are probability models? Probability models are mathematical constructions for describing different types of variability we can observe in real-world quantities. Suppose we are interested in the quantity X , which could be the outcome of a coin toss, rolling a die, or some measurement in industrial process. We refer to this kind of quantities as *random variables*, and use capital letters to denote them. Unlike a regular variable x that is a placeholder for a single value, the random variable X can have different *outcomes* every time it is observed. The possible outcomes of a coin toss are heads and tails. The possible outcomes of rolling a die are the numbers in the set $\{1, 2, 3, 4, 5, 6\}$. The outcomes of the industrial process are obtained by measuring the quantity of interest like the weight of loaf of bread.

A *probability model* is a way to characterize and quantify the variability of a random variable. We'll use the math notation $X \sim \mathcal{M}(\theta_X)$ to describe a random variable X that is distributed according to the probability model $\mathcal{M}(\theta_X)$. The math symbol " \sim " is a shorthand for the phrase "distributed according to." Whenever we talk about probability models in general, we'll use the calligraphic letters like \mathcal{M} to denote the *model family* and Greek letters like θ, α, μ , etc. to denote the model parameters.

Examples of probability model families include the discrete uniform family \mathcal{U}_d , the continuous uniform family \mathcal{U} , the normal family \mathcal{N} , etc. Each of these model families can be used to describe random variables whose distribution has a particular “shape.” Figure 2.1 shows six examples of random variables generated from six different probability models.

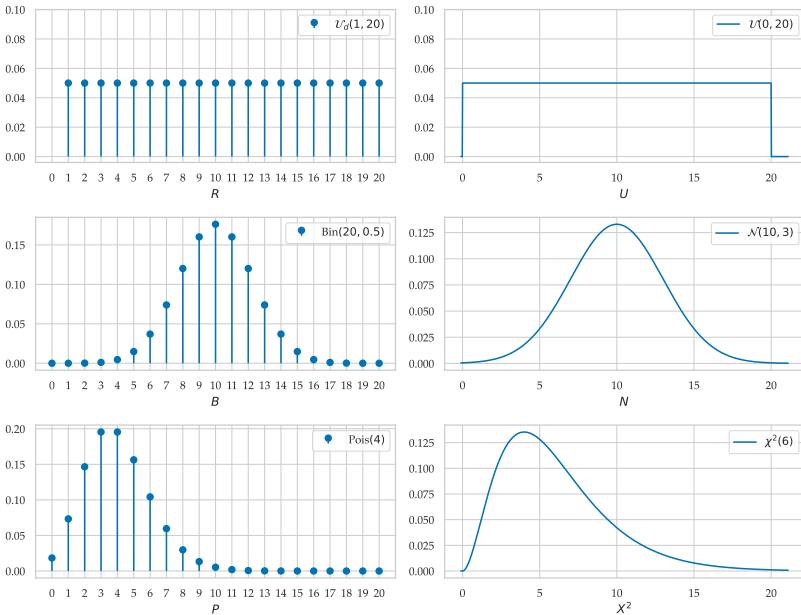


Figure 2.1: Six examples of probability models that describe the variability of six different random variables: R , U , B , N , P , and χ^2 . The three examples on the left side are *discrete* random variables, which can take on integer values. The three examples on the right are *continuous* random variables, which describe smoothly varying quantities.

Without going into too much details, here is a short description of the six random variables shown in Figure 2.1:

- $R \sim \mathcal{U}_d(1, 20)$ is a *discrete uniform* random variable whose outcomes are random integers in the range $\{1, 2, 3, \dots, 20\}$. All outcome have an equal probability of occurrence. You can think of the random variable R as rolling a 20-sided die.
- $U \sim \mathcal{U}(0, 20)$ is a *continuous uniform* random variable that assigns equal probability over the interval $[0, 20]$. The flatness of the curve tells us all values between 0 and 20 are equally likely to occur.
- $B \sim \text{Bin}(n = 20, p = 0.5)$ is a *binomial* random variable, which describes the counts of heads observed in $n = 20$ coin tosses of

a coin with probability of heads $p = 0.5$. We see the most likely outcome is to observe 10 heads, but the outcomes 8, 9, 11, and 12 also can occur with high probability.

- $N \sim \mathcal{N}(\mu = 10, \sigma = 3)$ is a *normal* random variable with parameters mean $\mu = 10$ and standard deviation $\sigma = 3$. The normal model is one of the most useful probability models that describes many real-world quantities.
- $P \sim \text{Pois}(\lambda = 4)$ is a *Poisson* random variable, which describes the number of occurrences of some event over a time period. For example, P could represent the number of phone calls you'll receive in one day. The parameter $\lambda = 4$ tells us that we can expect to receive four calls on average.
- $X^2 \sim \chi^2(\nu = 5)$ is a *chi-squared* random variable, which is used in statistics to describe sums of squared deviations of list of values from their expected values. The parameter ν describes the *degrees of freedom* and is related to the number of things summed together.

Later in this chapter, we'll describe each of these random variables in more details, and learn about their properties and applications for modelling real-world quantities (see sections 2.3 and 2.5). For the purpose of this introduction, our goal is to show some examples of the probability models. You can think of probability models as LEGOs that you can play with.

Example: kombucha bottling process Imagine that you work at a kombucha brewery and you're in charge of the bottling process. Your goal is to put exactly 1 litre (1000 ml) of kombucha in each bottle, but because of the bubbles in the kombucha, there is a natural variability in the total volume of kombucha that goes into each bottle. Figure 2.2 shows a scatter plot of the kombucha volume measurements from the last 500 bottles produced by your brewery.

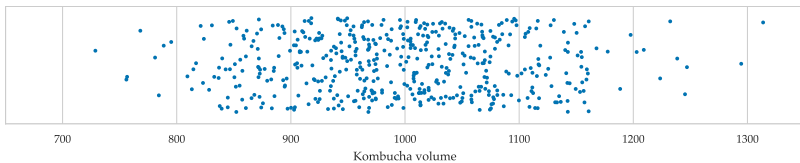


Figure 2.2: Observed values of the volume of Kombucha from 500 bottles. The average volume is 1000 ml with a standard deviation of 100 ml.

You choose to model the volume in each bottle as a normally distributed random variable $X \sim \mathcal{N}(\mu = 1000, \sigma = 100)$. The parameter $\mu = 1000$ determines the centre of the distribution. The

parameter $\sigma = 100$ describes the variability of the values around the centre. Figure 2.3 shows the model (a mathematical construct) for the random variable X , that models the variability of kombucha volume.

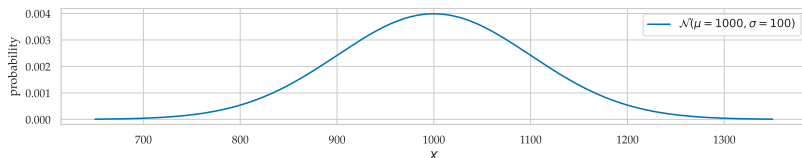


Figure 2.3: The random variable X distributed according to $\mathcal{N}(1000, 100)$. The probability of observing a value of X is proportional to the probability density (the height of the curve) that the model assigns to each value of X .

By choosing the appropriate parameters $\mu = 1000$ and $\sigma = 100$ for the normal model, you obtain a random variable X that approximates the variability observed in the kombucha volume. Note the math model assigns high probability to values in the interval $[800, 1200]$, and very low probability to values $X < 800$ and $X > 1200$. Compare the probability graph in Figure 2.3 with the observations in Figure 2.2 to convince yourself the normal model is a good fit.

It's important to understand the connection between the mathematical model shown in Figure 2.3 and the real-world kombucha values observations shown in Figure 2.2. The diagram in Figure 2.4 illustrates this “is a model of” relationship that exists between the real-world quantity (the volume of kombucha) and the math-world random variable $X \sim \mathcal{N}(1000, 100)$.

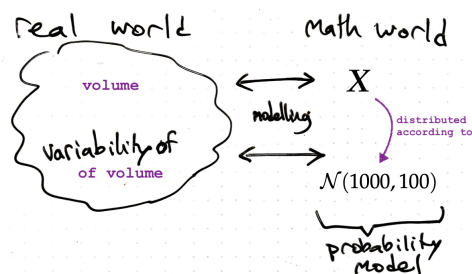


Figure 2.4: Modelling the volume of kombucha that goes into each bottle using the random variable $X \sim \mathcal{N}(1000, 100)$.

Using the math model $X \sim \mathcal{N}(1000, 100)$, you can do all kinds of probability calculations and predictions related to the kombucha bottling process. The options available to you when using the random variable X include: doing calculations based on math formulas, generating graphs and visualizations, and running computer

simulations. For example, you could estimate the probability of observing kombucha volumes $X < 800$, which is a outcome with business importance: distributors label all bottles containing less than 800 ml as defective and refuse to pay for them.

The goal of this example was to illustrate that probability models are not just abstract math constructs, but also very practical, since they help us model real-world quantities. We'll continue the discussion on this kombucha scenario in Section 2.5 (see page 142).

How are we going to do this? The goal of this chapter is to bring you to a deep understanding of probability theory concepts. We'll learn about probability models from various perspectives:

- **Math equations.** We'll give precise definitions of probability concepts expressed as math equations and formulas. Don't worry if your math skills are a little rusty: each piece of math notation will be translated into words, and each formula will be explained. Mathematical prerequisites will be presented in a just-in-time manner throughout the chapter, including calculus concepts (see Section 2.4).
- **Visualizations.** We can use graphical representations of probabilities to visualize random variables and probability models. Behind every equation, there is some "picture" you can draw to visualize what the equation is saying. You're already familiar with the tools we'll use for generating plots from the previous chapter (Matplotlib and Seaborn), so you'll find this easy.
- **Simulations.** Computer simulations allow us to generate observations from random variables, which is super useful for understanding their properties. The Python module `scipy.stats` contains pre-defined probability models for all the random variables we'll use in this chapter. Using a few lines of Python code you can simulate any random variable or process.

Each of these perspectives on probability theory will show you new ways to think about probabilities, and give you tools for working with random variables. Important concepts will be explained redundantly, using math, visually, and in computer code.

By the end of this chapter, you'll have gained valuable experience and intuition about probability theory, which will make your study of statistics in chapters ?? and ?? much easier and more interesting. Basically, in the next 200 pages you'll get to know the properties of probability models (the LEGOs for the XXIst century), so that we can play with them in the next two chapters.

Are you ready for this? Let's go.

The learning objectives for this chapter are:

- Understand the notion of a random variable X
- Recall key probability terminology and notation for random variables (pdf/pmf f_X , CDF F_X , expectation operator \mathbb{E}_X , etc.)
- Use computer models for random variables defined in the Python module `scipy.stats` for calculations.
- Calculate probabilities of events in discrete sample spaces using counting and combinatorics (permutations and combinations).
- Compute expectations of quantities that depend on random variables
- Compute the mean and the variance of probability distributions
- Calculate probabilities of events in continuous sample spaces using integration
- Determining CDF from the pdf of a continuous random variable
- Understand and compute integrals using multiple approaches: geometrically as areas under the graph of a function, using symbolic math (pen and paper or SymPy), and numerically (using functions in the module `scipy.integrate`)
- Calculate conditional probabilities and determine independence of events
- Describe the Gaussian distribution (properties, parameters, and intuition)
- Know the 10 most important probability distributions for statistics
- Generate samples of observations (x_1, x_2, \dots, x_n) from any random variable X
- Compute expectations on random samples (X_1, X_2, \dots, X_n) to obtain sampling distributions

2.1 Discrete random variables

Probability theory started when a bunch of mathematicians went to the casino and tried to use their math skills to compute what they can expect to win from different games of chance. Over time, probability theory was applied to many other situations where uncertainty plays a role, including biological processes, engineering, manufacturing, business, machine learning, and many other domains. Probability theory is the foundation of statistics.

In this section, we'll start our discussion about probability theory with the analysis of simple random variables like the outcomes of a coin toss and a die roll. We'll define some new concepts and terminology for describing random variables, give lots of examples, then talk about the calculations we can perform using random variables. We'll also show how to create computer models for random variables (Python functions and classes), and to use these computer models for probability calculations.

2.1.1 Definitions

Probabilities are real numbers between 0 and 1 that we assign to different events to describe how likely they are to occur. We denote the probability of event A as $\Pr(A)$. If an event has zero probability, $\Pr(A) = 0$, this means the event A never happens. At the other extreme, if $\Pr(A) = 1$, this means the event A is certain to occur. Probability values between 0 and 1 describe events that occur some of the time.

Random variables

A *random variable* is a mathematical model for describing unknown or uncertain quantities and assigning probabilities to their possible outcomes.

- *random variable* X : a quantity that takes on different values. We denote random variables by uppercase letters like X , Y , and Z .
- *sample space* \mathcal{X} : the calligraphic- X describes the set of all possible outcomes of the random variable X .
- *outcome*: the result of observing a random variable. We denote outcomes using lowercase letters like x , y , and z . A *simple* outcome describes one particular element of the sample space, like $X = c$, which can also be written as the set $\{X = c\}$. A *composite* outcome is a set of simple outcomes, like the set of numbers between a and b : $\{a \leq X \leq b\}$, or any other subset of the sample space.

- f_X : the *probability distribution function* is a function that assigns probabilities to the different outcomes in the sample space of the random variable X .

There are two kinds of random variables: discrete and continuous:

- A *discrete* random variable has a sample space \mathcal{X} that consists of discrete values, like the integers $\{0, 1, 2, 3, \dots\}$. See the left side of Figure 2.1 (page 2) for examples of discrete random variables.
- A *continuous* random variable has a continuous sample space \mathcal{X} , like the real numbers $\{0.1, 1.2, 3.14, -2.5, \dots\}$. Revisit the right side of Figure 2.1 to see examples of continuous random variables.

We use different kinds of “math machinery” to compute probabilities for these two types of random variables. In this section, we’ll focus on the definitions and formulas for discrete random variables, and defer the discussion on continuous random variables until Section 2.5.

Discrete random variables

Examples of discrete sample spaces include the outcomes of a coin toss $\{\text{heads}, \text{tails}\}$, the roll of a die $\{1, 2, 3, 4, 5, 6\}$, or a countably infinite set like the natural numbers $\mathcal{N} \stackrel{\text{def}}{=} \{0, 1, 2, 3, \dots\}$.

A discrete random variable X is described by a *probability mass function* f_X , which tells us how much probability “weight” is assigned to each of the possible outcomes:

$$\Pr(\{X = x\}) = f_X(x), \quad \text{for all } x \text{ in } \mathcal{X}.$$

The abbreviation *pmf* is often used to refer to the probability mass function.

An example of a probability mass function is shown in Figure 2.5.

Calculating probabilities We often need to compute the probabilities of composite outcomes, which we’ll describe as sets $\{a \leq X \leq b\} = \{a, a+1, a+2, \dots, b\}$. The probability of the composite outcome $\{a \leq X \leq b\}$ is obtained by computing the sum of the probability mass function for all values in the set:

$$\begin{aligned} \Pr(\{a \leq X \leq b\}) &= f_X(a) + f_X(a+1) + f_X(a+2) + \dots + f_X(b) \\ &= \sum_{x=a}^{x=b} f_X(x). \end{aligned}$$

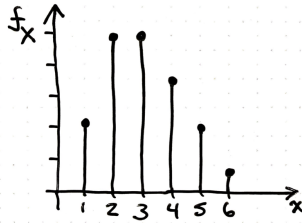


Figure 2.5: Illustration of the probability mass function f_X for the random variable X defined on the sample space $\mathcal{X} = \{1, 2, 3, 4, 5, 6\}$. The height of each line in the step plot tells us the probability of this outcome occurring.

Recall the symbol \sum (the capital Greek letter *sigma*) is the math notation for describing summations. The subscript of the summation $x = a$ tell us where to start the summation, and superscript $x = b$ tells when to end it. In words, we calculate the probability of a composite outcome by adding up the total probability of the outcomes it includes.

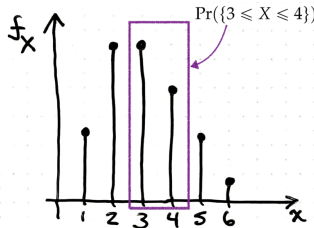


Figure 2.6: Illustration of the probability of the outcome $\{3 \leq X \leq 4\}$. The combined length of the two lines in the highlighted region correspond to the probability $\Pr(\{3 \leq X \leq 4\}) = \sum_{x=3}^{x=4} f_X(x)$.

The Figure 2.6 illustrates the calculation of the probability that the discrete random variable X will have outcome 3 or 4, which is the sum of the probability mass function values for these two outcomes: $\Pr(\{3 \leq X \leq 4\}) = f_X(3) + f_X(4) = \sum_{x=3}^{x=4} f_X(x)$.

Properties of probability mass functions Every probability mass function f_X satisfies the following math axioms:

- Nonnegativity: $f_X(x) \geq 0$ for all x in \mathcal{X} .
- Unit total: $\sum_{x \in \mathcal{X}} f_X(x) = 1$.

These two conditions are known as *Kolmogorov's axioms of probability*, in honour of the mathematician Andrey Kolmogorov who first introduced them. The first axiom states that probability functions

cannot take on negative values. The second axiom states that the total amount of probability over the whole sample space is 1. The symbol “ \in ” is math shorthand for “element of,” so the expression $\sum_{x \in \mathcal{X}} f_X(x)$ describes the summation of the function $f_X(x)$ over all values of x in the set \mathcal{X} .

We can derive some other properties of probability mass functions from the axioms. Define the outcome A to be an arbitrary subset of the sample space \mathcal{X} . The *numerical bound* tells us the probability of this outcome is less than or equal to one:

$$\Pr(A) \leq 1.$$

This follows from the unit-total axiom, which states $\Pr(\mathcal{X}) = 1$. Since A is a subset of the sample space \mathcal{X} , the probability of A can be at most 1.

For every outcome A , we can define its complement A^c , which consists of all outcomes of the sample space that are not in A . The *complement rule* tells us the probability of the complement A^c is equal to one minus the probability of the outcome A :

$$\Pr(A^c) = 1 - \Pr(A).$$

This rule follows from the definition of the complement A^c : together A and A^c make up the entire sample space \mathcal{X} . Using the *unit total* axiom, we therefore have $\Pr(A) + \Pr(A^c) = 1$, which we can rearrange to obtain the complement rule.

Example 1: coin toss Consider the random variable C that describes the outcome of a coin toss for a balanced coin. The sample space for the random variable C has two possible outcomes: {heads, tails}. Since we assume the coin is balanced, the two outcomes have equal chance of occurring. The probability mass function that describes this random variable is

$$f_C(\text{heads}) = 0.5 \quad \text{and} \quad f_C(\text{tails}) = 0.5.$$

The values of the function f_C are shown in Figure 2.7.

Example 2: rolling a six-sided die The random result of rolling a six-sided die can be described as a random variable D over the sample space $\{1, 2, 3, 4, 5, 6\}$. If we assume the die is fair, then each of the outcomes will have an equal probability of occurring, so the probability mass function f_D has the following values:

$$f_D(1) = \frac{1}{6}, f_D(2) = \frac{1}{6}, f_D(3) = \frac{1}{6}, f_D(4) = \frac{1}{6}, f_D(5) = \frac{1}{6}, f_D(6) = \frac{1}{6}.$$

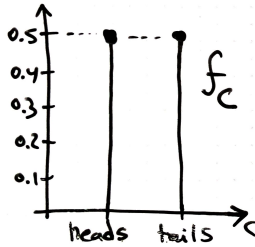


Figure 2.7: The probability mass function of the random variable C , which describes a coin toss. The two possible outcomes are $\{\text{heads}, \text{tails}\}$, and f_C assigns equal probability $\frac{1}{2}$ to each outcome.

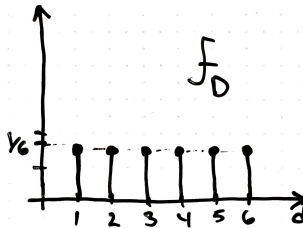


Figure 2.8: The probability mass function of the random variable D , which describes a die roll with six possible outcomes $\{1, 2, 3, 4, 5, 6\}$. Note the probability mass function is defined only for a discrete set of inputs that correspond to the possible outcomes in the sample space.

A plot of the probability mass function for the random variable D is shown in Figure 2.8.

It can be very useful to encode the mathematical definition of the probability mass function f_D as a Python function `fD`. This will allow us to do probability calculations involving the random variable D very quickly. The Python function that corresponds to the probability mass function f_D is defined below.

```
>>> def fD(d):
    if d in {1,2,3,4,5,6}:
        return 1/6
    else:
        return 0
```

code
2.1.1

This function returns the value $\frac{1}{6} = 0.16666\dots$ whenever the input d is one of the numbers in the sample space $\{1, 2, 3, 4, 5, 6\}$, and returns zero otherwise.

If this is the first time you're seeing the definition of a Python function, you might pause your reading here and jump to the Python tutorial in Appendix C, where you can learn about the Python syntax of `def` statements, which we use to define Python functions.

In order to call the function `fD` we just defined, we use the

function name followed by the arguments specified in parentheses:

```
>>> fD(3)
0.16666666666666666
```

code
2.1.2

Note the Python syntax for calling functions is the same as the math notation for evaluating the function f_D for the input $d = 3$, $f_D(3)$.

Example 3: hard disk failures We can model the number of hard disk failures expected to occur in a given data centre as a random variable H . The sample space for the random variable H is $\{0, 1, 2, 3, \dots\}$, since the number of hard disk failures can be a nonnegative integer (a count). We'll denote $\{H = h\}$ the outcome “ h hard disks failed this month.” Based on the hard disk manufacturer's data sheets and the specifics of the data centre, we decide to model the number of hard disk failures using a *Poisson* distribution with parameter $\lambda = 20$:

$$H \sim \text{Pois}(\lambda = 20).$$

Recall the symbol “ \sim ” is shorthand for the phrase “is distributed according to,” so the above definition tells us H is distributed according to the Poisson model with parameter $\lambda = 20$. The probability mass function that corresponds to the Poisson model with parameter λ is given by the following equation:

$$f_H(h) = \frac{\lambda^h e^{-\lambda}}{h!}, \text{ for } h \in \{0, 1, 2, 3, \dots\},$$

where e^x is the exponential function and $n!$ is the factorial function. The Poisson model corresponds to a whole family of different distributions that depend on the choice of parameter λ . We created the specific model for our situation by choosing the parameter $\lambda = 20$. Figure 2.9 illustrates the possible outcomes we can expect to occur for a Poisson model with parameter $\lambda = 20$ (twenty hard disk failures on average).

Using the formula for the probability mass function f_H , we can calculate the probability of any outcome, like $\Pr(\{H = 23\}) = \frac{20^{23} e^{-20}}{23!} = 0.06688$, or intervals of outcomes $\Pr(\{18 \leq H \leq 22\}) = f_H(18) + f_H(19) + f_H(20) + f_H(21) + f_H(22) = \sum_{h=18}^{22} f_H(h) = 0.4236$. It's possible to do these probability calculations using pen and paper, but it is much easier to do them using the computer, as we'll now see.

First, we need to define a Python function `fH` for computing the probability mass function f_H . We can call the functions `np.exp` and `np.math.factorial` from the NumPy module to do the e^x and $n!$ calculations for us.

code
2.1.3

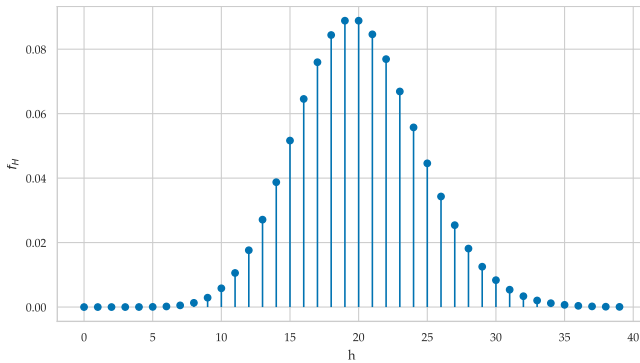


Figure 2.9: Graphical representation of the random variable H distributed according to the Poisson model with parameter $\lambda = 20$. We see the most likely values of h are around 20, but all outcomes from 15 to 25 have significant probability of occurring. Even extreme outcomes like 10 and 30 can occur, with small probability.

```
>>> import numpy as np
>>> def fH(h):
    lam = 20
    return lam**h * np.exp(-lam) / np.math.factorial(h)
```

Note the complicated-looking code inside the Python function `fH` matches exactly the complicated-looking math expression for the function f_H that we saw above. We'll use this pattern repeatedly in the remainder of the book, converting math expressions into code expressions so that we do computations with them quickly.

We can use the function `fH` to compute the probability of simple outcomes like $\{H = 23\}$:

```
>>> fH(23)
0.06688147366240181
```

code
2.1.4

The probability of a composite outcome is the sum of the probabilities of the individual elements. To compute the probability of the composite outcome $\{18 \leq H \leq 22\}$, we use the following Python expression:

```
>>> sum([fH(h) for h in range(18,22+1)])
0.42358294520135187
```

code
2.1.5

The Python function `range(a,b)` is equivalent to the list of values $[a, a+1, a+2, \dots, b-1]$, which does not include the upper limit b . If we want a list of values that includes b we must call `range(a,b+1)`, which produces the list $[a, a+1, a+2, \dots, b-1, b]$. You'll see this $+1$ added in numerous code examples below, whenever we want to create a list up-to-and-including some upper limit.

Recall Kolmogorov's second axiom, which requires that the sum of f_H over the entire sample space must be one: $\sum_{h=0}^{h=\infty} f_H(h) = 1$. The code below shows how we can verify that the Python function `fH` satisfies Kolmogorov's axiom:

```
>>> sum([fH(h) for h in range(0,100+1)])
1.0
```

code
2.1.6

Note we calculated the summation only until $h = 100$ and not until $h = \infty$. In general, it's not possible to do summation until infinity on computers. Stopping the summation at $h = 100$ is okay in this case, because the probability values $f_H(101)$, $f_H(102)$, etc. are very small numbers, which are negligible when compared to the answer 1.0.

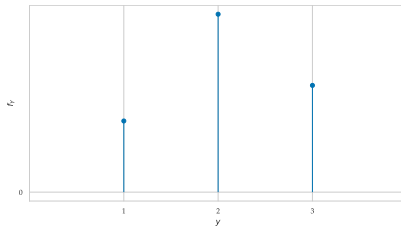
Note the probability distributions f_C and f_D from the earlier examples also satisfy Kolmogorov's second axiom (unit total probability): $f_C(\text{heads}) + f_C(\text{tails}) = 1$ and $f_D(1) + f_D(2) + f_D(3) + f_D(4) + f_D(5) + f_D(6) = 1$.

Exercises

E2.1 Compute the probability of the outcome $\{2 \leq D \leq 5\}$ for the random variable D defined in Example 2.

E2.2 Consider the random variable D_4 that describes the outcome of rolling a tetrahedral (four-sided) die. The sample space is $\{1, 2, 3, 4\}$. Assuming the die is fair, what is the probability mass function for the random variable D_4 ?

E2.3 The random variable Y is defined over the sample space $\{1, 2, 3\}$. A colleague published a graph of the probability mass function f_Y , but forgot to label the vertical axis. Can you find a way to compute the probability value $f_Y(3)$ from the graph?



Hint: Use a ruler to measure the lengths of the stems.

E2.4 Use the *complement rule* to calculate the probability $\Pr(\{5, 6\})$ for the die roll random variable D , given that $\Pr(\{1, 2, 3, 4\}) = \frac{4}{6}$.

2.1.2 Cumulative distribution functions

The *cumulative distribution function* F_X of the random variable X describes the probability of outcomes that are smaller than or equal to some value b :

$$F_X(b) \stackrel{\text{def}}{=} \Pr(\{X \leq b\}).$$

The cumulative distribution function of the random variable X is computed by adding up all the values of the probability mass function $f_X(x)$ until $x = b$. See Figure 2.10 for an illustration.

For example, we can define a Python function `FH` that computes the cumulative distribution function F_H for the random variable $H \sim \text{Pois}(\lambda = 20)$ from Example 3. Recall we previously defined a Python function `fH` that computes the probability mass function f_H in code 2.1.3. Following the definition above, we can therefore define $F_H(b)$ as the sum of the values of f_H until $h = b$, as shown below.

```
>>> def FH(b):  
        return sum([fH(h) for h in range(0,b+1)])
```

code
2.1.7

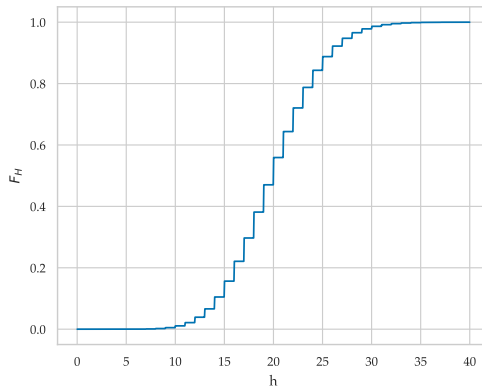


Figure 2.10: Graph of the CDF function $F_H(h)$ for the random variable H from Example 3. Note the graph of F_H increases in jumps at each value of h , and the height of each jump at h is equal to the value of the probability mass function $f_H(h)$.

Intuitively speaking, the cumulative distribution function corresponds to a pre-computed summation of the values of the probability mass function f_X . Knowing the cumulative distribution function F_X for the random variable X gives us an alternative way to compute probabilities. We can find the probability of outcomes between a and b by computing the difference in the value of the cumulative distribution function:

$$\Pr(\{a \leq X \leq b\}) = F_X(b) - F_X(a - 1).$$

The value $F_X(b)$ contains the pre-computed sum of f_X from $x = 0$ until $x = b$, so if we want to know the sum of the values of f_X from $x = a$ to $x = b$, we can start with $F_X(b)$ and subtract $F_X(a - 1)$, which is the sum of f_X until $x = a - 1$.

For example, if we wanted to compute the probability of a composite outcome $\{18 \leq H \leq 22\}$, we can simply compute the change in the value of the Python function FH we defined in code 2.1.7.

```
>>> FH(22) - FH(18-1)
0.42358294520135187
```

code
2.1.8

The value we obtain is the same as the probability we calculated using the summation in code 2.1.5.

The cumulative distribution function has the following properties:

- If $b_1 < b_2$ then $F_X(b_1) \leq F_X(b_2)$, which means the function F_X is *nondecreasing*.
- $F_X(b) \leq 1$ for all b
- $\Pr(\{X > b\}) = 1 - F(b)$ for all b

The first property follows from the *nonnegativity* property of the probability mass function $f_X(x) \geq 0$ for all $x \in \mathcal{X}$. The second property follows from the *unit total* property of the probability density function $\sum_{x \in \mathcal{X}} f_X(x) = 1$. The third property is a consequence of the complement rule $\Pr(A^c) = 1 - \Pr(A)$, since the outcome $\{X > b\}$ is the complement of the outcome $\{X \leq b\}$.

The functions F_X and f_X contain the same information. The values of F_X are simply the cumulative sums of f_X , and we can obtain f_X from F_X by looking at the lengths of the jumps in the graph of F_X , for each of the values $x \in \mathcal{X}$.

Inverse of the cumulative distribution function

The inverse function of the cumulative distribution function is denoted $F_X^{-1}(q)$. The inverse-CDF is also sometimes called the *percentile point function* or *quantile function*, since it tells us the positions of the quantiles of the random variable X . The value $F_X^{-1}(q) = x_q$ tells how far you need to go in the sample space so that the outcome $\{X \leq x_q\}$ contains at least a proportion q of the total probability: $F_X(x_q) \geq q$. For example, the first quartile (25% percentile or $q = 0.25$ quantile) is the value of $x_{0.25}$ when the cumulative distribution reaches the value 0.25, $F_X(x_{0.25}) \geq 0.25$.

We use the point percentile function to compute confidence intervals that contain a certain percentage of the probability mass. For

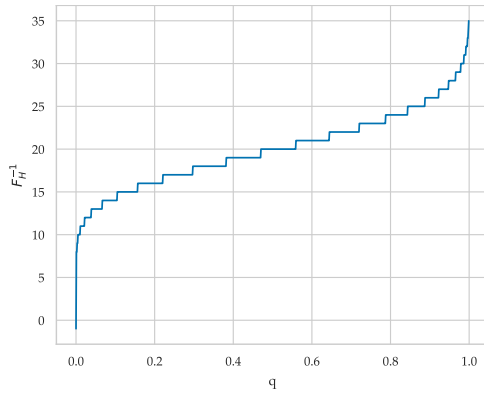


Figure 2.11: Graph of the inverse-CDF function $F_H^{-1}(q)$ for the random variable H , which contains the same information as the CDF function F_X .

example, the interval between $F_X^{-1}(0.025)$ and $F_X^{-1}(0.975)$ contains 95% of the probability density of the random variable X .

Note that the functions F_X and F_X^{-1} contain the same information, but we have two functions because we need to “query” this information in both directions. In the forward direction, we specify some value $b \in \mathcal{X}$ and the function $F_X(b)$ tells you the value $\Pr(\{X \leq b\})$. In the inverse direction, we’re starting from some proportion $q \in [0, 1]$ of the total probability, and we want to know the smallest value x_q such that $\Pr(\{X \leq x_q\}) \geq q$.

Exercises

E2.5 Calculate $\Pr(\{10 \leq H \leq 30\})$ using the Python function `FH`, which computes cumulative distribution function F_H for the random variable H introduced in Example 3.

E2.6 Consider the random variable D , which describes the roll of a fair die we saw in Example 2. Compute the following six values of the cumulative distribution function: $F_D(1)$, $F_D(2)$, $F_D(3)$, $F_D(4)$, $F_D(5)$, $F_D(6)$.

E2.7 Draw the graph of the cumulative distribution function F_D by hand using the values obtained in the previous exercise.

E2.8 Compute the median of the random variable H which corresponds to the value $F_H^{-1}(\frac{1}{2})$.

Hint: The answer is the smallest value of b such that $F_H(b) \geq 0.5$.

Hint: Evaluate the Python function `FH` for the inputs $h = 18$, $h = 19$, $h = 20$, $h = 21$, $h = 22$ to find the first value of h such that `FH(h)` is

greater than 0.5.

E2.9 Compute the position of the first quartile $F_H^{-1}(0.25)$ for the random variable H .

Hint: Use the same strategy as in the previous exercise.

2.1.3 Expected value calculations

Suppose we're interested in calculating some value $w(X)$, which depends on the random variable X . The function $w: \mathcal{X} \rightarrow \mathbb{R}$ assigns different “winning” amounts to the different outcomes of the random variable X . Since the input to the function w is a random variable, the output value $w(X)$ is also a random variable. The *expected value* of $w(X)$ is obtained by computing the average over all the possible outcomes x of the random variable X , which corresponds to the following summation:

$$\mathbb{E}_X[w(X)] \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} w(x) f_X(x).$$

The expected value “weights” each value of $w(x)$ by the probability of the outcome $\{X = x\}$, summed over all possible outcomes for the random variable X . The math shorthand symbol \mathbb{E}_X describes the expected value of a quantity that depends on the random variable X .

The name *expected value* is appropriate for this calculation: the value of $w(X)$ will take on different values depending on the random variable X , but $\mathbb{E}_X[w(X)]$ tells us the expected value of w on average, under the randomness of X .

Example 4 Suppose someone offers you to play a game of chance with a six-sided die. It costs \$1 to play one round of this game. You win \$5 if you roll a 6 and you win nothing if you roll any other number. In other words, the winnings function w for this game is

$$w(1) = w(2) = w(3) = w(4) = w(5) = \$0 \quad \text{and} \quad w(6) = \$5.$$

Is it worth playing this game?

We assume the die is fair, and it is described by the random variable D with distribution $f_D(d) = \frac{1}{6}$, for all d in the sample space $\{1, 2, 3, 4, 5, 6\}$. We then calculate the expected winning from this

game like so:

$$\begin{aligned}
 \mathbb{E}_D[w(D)] &= \sum_{d=1}^{d=6} w(d) f_D(d) \\
 &= w(1)\frac{1}{6} + w(2)\frac{1}{6} + w(3)\frac{1}{6} + w(4)\frac{1}{6} + w(5)\frac{1}{6} + w(6)\frac{1}{6} \\
 &= (\$0)\frac{1}{6} + (\$0)\frac{1}{6} + (\$0)\frac{1}{6} + (\$0)\frac{1}{6} + (\$0)\frac{1}{6} + (\$5)\frac{1}{6} \\
 &= \frac{\$5}{6} \approx 83 \text{ cents.}
 \end{aligned}$$

Since the expected winning from this game is smaller than the cost of playing, we conclude that this game is not worth playing. We're losing 17 cents on average in each round the game! We might as well throw our money down the drain—it would be just as efficient.

Let's now see how to compute the expected value $\mathbb{E}_D[w(D)]$ using computer code. First we define the winning function $w(d)$ that returns the dollar amount we obtain depending on the outcome d of the die roll.

```
>>> def w(d):
    if d == 6:
        return 5
    else:
        return 0
```

code
2.1.9

We previously defined the Python function fD in code 2.1.3 above. We can now compute the expected value by summing the expression $w(d)*fD(d)$ for values of d ranging from 1 to 6.

```
>>> sum([w(d)*fD(d) for d in range(1,6+1)])
0.8333333333333333
```

code
2.1.10

The answer we obtain from the Python calculation is the same as the answer we obtained for $\mathbb{E}_D[w(D)]$ using the math calculations.

Computing expectations is crucial for many probability applications. Indeed, we could say that computing expected values is one of the main tools we have for dealing with uncertainty. We can't predict the value of $w(X)$ since X is a random variable, but we can average over all the possible outcomes of X to calculate the expected value $\mathbb{E}_X[w(X)]$, which takes into account the probability of all possible outcomes.

Expectations are also used to define two important “quantities of interest” for any random variable: its mean and its variance, which we'll discuss next.

Measures of centre and dispersion

The *mean* μ_X and the *variance* σ_X^2 are two properties of any random variable X that capture important aspects of its behaviour. The mean and the variance are defined as expectation calculations.

The mean of the discrete random variable X with probability mass function f_X is defined as:

$$\mu_X \stackrel{\text{def}}{=} \mathbb{E}_X[X] = \sum_{x \in \mathcal{X}} x \cdot f_X(x).$$

The mean is a single number that tells us what value we can expect to obtain on average from the random variable X . The mean is also called the *average* or the *expected value* of the random variable X .

The variance of a discrete random variable is defined as follows:

$$\sigma_X^2 \stackrel{\text{def}}{=} \mathbb{E}_X[(X - \mu_X)^2] = \sum_{x \in \mathcal{X}} (x - \mu_X)^2 \cdot f_X(x).$$

The variance formula calculates the average squared deviation of the random variable X from its mean μ_X , which is a measure of the dispersion of the distribution. A small variance indicates the outcomes of X are tightly clustered near the mean μ_X , while a large variance indicates the outcomes of X are widely spread. The square root of the variance is the standard deviation $\sigma_X = \sqrt{\sigma_X^2}$.

We've seen these concepts of mean, variance, and standard deviation previously in Section ?? when we computed the mean, variance, and standard deviation descriptive statistics of different variables in a dataset. The purpose of the calculations is the same—to find the average value and the dispersion of the values—but instead of working with data values observed in a particular dataset, we're making a theoretical calculation over all possible outcomes of the probability distribution.

Let's see how to calculate the mean and variance of the random variables D and H we introduced earlier in examples 2 and 3.

Example 5 Recall the random variable D that describes the outcome of rolling a six-sided die, which has a probability mass function $f_D(d) = \frac{1}{6}$, for $d \in \{1, 2, 3, 4, 5, 6\}$. The mean of this distribution is computed as follows:

$$\begin{aligned} \mu_D \stackrel{\text{def}}{=} \mathbb{E}_D[D] &= \sum_{d=1}^{d=6} d \cdot f_D(d) \\ &= 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} \\ &= \frac{21}{6} = 3.5. \end{aligned}$$

This means the average value of rolling a six sided die is 3.5.

The code example below shows how to perform the same computation using Python, based on the probability mass function `fD` defined in code 2.1.1.

```
>>> sum([d*fD(d) for d in range(1,6+1)])
3.5
```

The formula for the variance of D is

$$\sigma_D^2 = \mathbb{E}_D[(D - \mu_D)^2] = \sum_{d=1}^6 (d - 3.5)^2 \cdot f_D(d).$$

We can easily carry out this calculation in Python as follows:

```
>>> sum([(d-3.5)**2 * fD(d) for d in range(1,6+1)])
2.9166666666666665
```

code
2.1.12

Example 6 Let's now compute the mean and the variance of the random variable $H \sim \text{Pois}(\lambda = 20)$, which describes the number of hard disk failures. The formula for the mean is

$$\mu_H \stackrel{\text{def}}{=} \mathbb{E}_H[H] = \sum_{h=0}^{\infty} h \cdot f_H(h).$$

Note the summation goes all the way to infinity, but the probability mass function $f_H(h)$ is negligibly small when h is large, so the summation until $h = 100$ gives the same result as the summation until infinity. We already defined the probability mass function `fD` in code 2.1.3, so we can jump straight to the $\mathbb{E}_H[H]$ calculation:

```
>>> sum([h*fH(h) for h in range(0,100+1)])
20.0
```

code
2.1.13

If you ever had doubts about the usefulness of computers for doing probability calculations, you'll surely agree this is a good occasion to use a computer, since otherwise you'd have to compute a summation with 101 terms using pen and paper!

The formula for the variance of H is

$$\sigma_H^2 = \mathbb{E}_H[(H - \mu_H)^2] = \sum_{h=0}^{\infty} (h - 20)^2 \cdot f_H(h),$$

and the Python code for computing the variance is shown below.

```
>>> sum([(h-20)**2 * fH(h) for h in range(0,100+1)])
20.0
```

code
2.1.14

Recall that the standard deviation σ_H is defined as the square root of the variance σ_H^2 . We can use the square-root function from the `numpy` package to compute the standard deviation:

```
>>> import numpy as np
>>> np.sqrt(20)
4.47213595499958
```

code
2.1.15

Intuitively speaking, this result tells us the “width” of the distribution f_H is approximately 5, and the most likely outcomes will be in the interval $[\mu_H - \sigma_H, \mu_H + \sigma_H] = [15, 25]$. As the operator of the data centre, this is really useful to know when planning.

Expectation formulas

To get a better feeling of the properties of the expectation operator, consider the following general rules that apply for all expectation calculations:

- The expected value of a constant is the constant itself:

$$\mathbb{E}_X[c] = c.$$

- The expected value of mX is m times the expected value of X :

$$\mathbb{E}_X[mX] = m\mathbb{E}_X[X].$$

- Linearity of expectation:

$$\mathbb{E}_X[\alpha g(X) + \beta h(X)] = \alpha \mathbb{E}_X[g(X)] + \beta \mathbb{E}_X[h(X)].$$

These formulas will come in handy when evaluating expectations of random variables that are defined in terms of other random variables.

Variance formulas

Let's now define a function $\mathbf{var}(X)$ that computes the variance of the random variable X :

$$\mathbf{var}(X) \stackrel{\text{def}}{=} \mathbb{E}_X[(X - \mu_X)^2].$$

This will allow us to understand the properties of variance.

- The variance of a constant is zero:

$$\mathbf{var}(c) = 0.$$

- The variance of the variable $X + b$ is the same as the variance of the variable X :

$$\mathbf{var}(X + b) = \mathbf{var}(X).$$

- The variance of the expression mX is m^2 times the variance of the variable X :

$$\mathbf{var}(mX) = m^2 \mathbf{var}(X).$$

There is another useful formula for computing the variance of a random variable in terms of the expectation of $\mathbb{E}_X[X^2]$ and $\mathbb{E}_X[X]$:

$$\begin{aligned}\mathbb{E}_X[(X - \mu_X)^2] &= \mathbb{E}_X[X^2] - \mu_X^2 \\ &= \sum_{x \in X} x^2 f_X(x) - \mu_X^2.\end{aligned}$$

This is known as the “parallel axis theorem” in physics, but doesn’t have a name in probability theory. People just call it the “simple” formula for the variance, since it is easier to compute compared to $\mathbb{E}_X[(X - \mu_X)^2]$. I’ll ask you to derive this formula in problem P2.7.

Example 7 Consider the random variable $Y = mX + b$, which is defined as a function of the random variable X . The random variable Y describes the same underlying random events as the random variable X , but assign different values to the outcomes.

Let’s use the expectation formulas we showed above to see how the mean of the random variable Y is related to the mean of the random variable X :

$$\mu_Y = \mathbb{E}_X[mX + b] = m\mu_X + b,$$

which follows from the linearity of the expectation operator. The mean is transformed exactly like the values of the random variable.

Now let’s compute the variance of $Y = mX + b$:

$$\mathbf{var}(Y) = \mathbf{var}(mX + b) = \mathbf{var}(mX) = m^2 \mathbf{var}(X).$$

The second equation is true because the variance calculation “ignores” the additive constant b . The third equation holds because the multiplicative scaling constant m has a squared effect when computing the variance.

Exercises

E2.10 Consider the gambling game involving a coin toss random variable C with winning function $w(c)$ defined as $w(\text{heads}) = \$1.90$ and $w(\text{heads}) = \$0$. Calculate the expected value $\mathbb{E}_C[w(C)]$.

E2.11 Compute $\mathbb{E}_D[u(d)]$ where $u(5) = \$3$, $u(6) = \$4$, and $u(d) = \$0$ for $d \in \{1, 2, 3, 4\}$.

E2.12 Consider the random variable $Y = mX + b$. Show that $\sigma_Y^2 = m^2 \sigma_X^2$ by starting from the definition $\sigma_Y^2 = \mathbb{E}_X[(Y - \mu_Y)^2]$.

E2.13 Compute the expected value of the random variable D_{20} , which describes the outcome of rolling a 20-sided die with distribution $f_{D_{20}}(d) = \frac{1}{20}$ for all d in $\{1, 2, 3, \dots, 20\}$.

2.1.4 Computer models for random variables

Computers are very useful for doing probability calculations. We saw in the above code examples how to define Python functions `fC`, `fD`, `fH`, which correspond to the probability mass functions f_C , f_D , f_H of the random variables C , D , H . But we don't have to create all our computer models by ourselves! The Python module `scipy.stats` contains pre-defined code for all the probability distributions that we'll use in this book. See the table on page ?? in Appendix ?? for a complete list of the distributions.

This means, if you need to use a probability model for the Poisson distribution in some calculations, you just need to import the `poisson` model using a command like `from scipy.stats import poisson`, then create a Poisson random variable object `rvX` by initializing the `poisson` model with your choice of parameters. (we'll explain how exactly to create `rvX` in the code examples below). You can then use the methods `rvX.pmf(x)` to obtain the value of the probability mass function $f_X(x)$, or `rvX.cdf(b)` to obtain the values of the CDF $F_X(b)$.

In this section, we'll discuss the affordances of the computer models defined in `scipy.stats`. What calculations are possible? We'll start by listing the most important methods available on the object `rvX`, relying on the code defined in the module `scipy.stats`.

- `rvX.pmf(x)` $\stackrel{\text{def}}{=} f_X(x)$: the probability mass function f_X . We can use this method to calculate the probability of any composite outcome as the sum over a range of inputs: $\Pr(\{a \leq X \leq b\}) = \text{sum}([\text{rvX.pmf}(x) \text{ for } x \text{ in range}(a, b+1)])$.
- `rvX.cdf(b)` $\stackrel{\text{def}}{=} F_X(b)$: the cumulative distribution function corresponds to the probability $\Pr(\{X \leq b\})$. We can compute the probability of the outcome $\{a \leq X \leq b\}$ by calculating $F_X(b) - F_X(a-1) = \text{rvX.cdf}(b) - \text{rvX.cdf}(a-1)$.
- `rvX.ppf(q)` $\stackrel{\text{def}}{=} F_X^{-1}(q)$: the *percent point function* is the inverse of the CDF function. We specify the lower tail probability q as the input, and the output of `rvX.ppf(q)` tells us the smallest value x_q such that $\Pr(\{X \leq x_q\})$ contains at least q of the total probability.
- The object `rvX` has numerous methods for obtaining the properties of the random variable like its mean, median, mode, variance, standard deviation, and others:

- ▷ `rvX.mean()` $\stackrel{\text{def}}{=} \mu_X$: the mean of the distribution
- ▷ `rvX.var()` $\stackrel{\text{def}}{=} \sigma_X^2$: the variance of the distribution

- ▷ `rvX.std()` $\stackrel{\text{def}}{=} \sigma_X$: the standard deviation of the distribution. Recall that the standard deviation is the square root of the variance: $\sigma_X \stackrel{\text{def}}{=} \sqrt{\sigma_X^2}$.
- ▷ `rvX.median()` $\stackrel{\text{def}}{=} F_X^{-1}(\frac{1}{2})$: the median of the distribution
- ▷ `rvX.support()` $\stackrel{\text{def}}{=} \mathcal{X}$: the sample space of the random variable
- `rvX.interval(α)` $\stackrel{\text{def}}{=} [F_X^{-1}(\frac{\alpha}{2}), F_X^{-1}(1 - \frac{\alpha}{2})]$: the interval method computes a two-sided *confidence interval* (CI) that contains $(1 - \alpha)$ of the total probability of the random variable X .
- `rvX.rvs(size= n)`: generate a sequence of n observations from the random variable X , which we denote (x_1, x_2, \dots, x_n) . Such computer-generated observations are useful for visualizing the variability we can expect to observe when we generate random samples from a distribution, which is useful if you want to model the variability of random samples collected from a population.
- `rvX.expect(w)` $\stackrel{\text{def}}{=} \mathbb{E}_X[w(X)]$: computes the expected value of the function w with respect to the distribution of the random variable X . Recall that $\mathbb{E}_X[w(X)] = \sum_{x \in \mathcal{X}} w(x) \cdot f_X(x)$, so the method `rvX.expect(w)` is equivalent to calculating

`sum([w(x)*rvX.pmf(x) for x in range(xmin,xmax+1)]),`

where `xmin` is the smallest value in \mathcal{X} and `xmax` is the largest value in \mathcal{X} .

To summarize, what I’m saying is that anything you might want to know about the random variable X is available at the tip of your fingers once you create the computer model `rvX`. The module `scipy.stats` has predefined computer models for all the important distributions in statistics: `poisson`, `randint`, `bernoulli`, `binom`, `uniform`, `norm`, `t`, `beta`, etc. You’ll learn all about these distributions later in this chapter (Section 2.3 for discrete distributions and Section 2.6 for continuous distributions). You can think of these probability models as different kinds of LEGO blocks available for you to play with.

In the next section, we’ll show how to create the computer model `rvH` for the hard-disks failures random variable H .

2.1.5 Hard disks example

Suppose you’re the operator of a data centre, and you want to estimate the number of hard disk failures that will occur this month.

Based on the specifics of your data centre, you know that 20 hard disk failures occur on average each month, but knowing the average is not sufficient for the types of questions you need to answer. What is the probability of observing 21 hard disk failures? What is the probability of having 25 hard disk failures or less? Can you give a range of outcomes that will occur 95% of the time? This is the types of questions your colleagues are interested in. The software engineering department needs to know the probabilities of different outcomes to design the redundant storage system. The legal department wants to know “worst case” scenarios to write the *Service Level Agreement* documents. The finance people are interested in estimating costs of replacement disks.

All these requests for estimates are piling up in your inbox, but despite your interest in data science topics, you never seem to find the time to do the probabilistic modelling exercise needed to answer these questions. One day during a high-level meeting, your colleagues decide to gang up on you and complain loudly about the lack of estimates, and put you on the spot in front of everyone.

You decide to get this done right then and there, and tell people:

“Relax everyone, we can do these estimates right now using Python.”

Everyone seems immediately reassured.

You know the Poisson family of probability models is well suited for describing the random number of hard disk failures in general. To obtain a random variable `rvH` that has the desired distribution, you can initialize the Poisson model with the parameter $\lambda = 20$.

You proceed to share your screen so everyone can see, open a Python shell, and start typing

```
>>> from scipy.stats import poisson
>>> rvH = poisson(20)
```

code
2.1.16

The code above imports the `poisson` model from the SciPy package `scipy.stats` and creates an instance of it called `rvH` initialized with parameter $\lambda = 20$. Now that you have the computer model `rvH`, you can use all the methods like `rvH.pmf(h)`, `rvH.cdf(b)`, `rvH.ppf(q)`, `rvH.rvs(n)`, etc. to do calculations with the random variable H . Feeling reassured by the plethora of methods available to you, you explain what you want from your colleagues during the meeting: “I want you to give me any probability question related to hard disk failure rates now, and I’ll use the probability model to answer your question to the best of my ability. Live.”

There is a moment of silence in the room as people are processing your directives. You decide to use the time to compute the probability of some outcomes:

code
2.1.17

```
>>> rvH.pmf(20)
0.0888353173920848
>>> rvH.pmf(21)
0.08460506418293791
>>> rvH.pmf(22)
0.07691369471176195
```

You explain to your colleagues this means the probability of observing 20 failures next month is 8.88%, the probability of observing 21 failures is 8.46%, and the probability of 22 failures is 7.69%.

Alice from accounting interrupts with a question. “Wait, I thought you said the expected value is 20. Now you’re telling us there is just 8% chance of that happening?”

“Yes, the average is $\mu_H = 20$, but we could have 21, 22, 23, or any other number of failures next month.”

“So we can’t know for sure how many failures will occur?”

“No, we can’t know for sure since failures are random, but we can think about the different possible outcomes and plan accordingly. For example, we could run simulations to—.” You stop yourself mid-sentence because you sense this meeting can go on forever if you start explaining each concept in detail. Better show than tell.

In order to better describe the range of values for the random variable H , you compute the two important statistics of the probability distribution:

```
>>> rvH.mean()
20.0
>>> rvH.std()
4.47213595499958
```

code
2.1.18

You interpret these numbers for your colleagues by saying: “This means that we can expect roughly 20 plus or minus 5 failures on average.”

“What do you mean ‘plus or minus 5’?” asks Bob from sales.

“I mean that the number of failures will likely be between $20 - 5 = 15$ and $20 + 5 = 25$.” You then proceed to compute the exact probability by summing the probabilities of the individual outcomes in that range.

```
>>> sum([rvH.pmf(h) for h in range(15,25+1)])
0.782950746174042
```

code
2.1.19

In other words, 78.2% of data centres like ours will experience between 15 and 25 failures.

You then say “Here is a plot that shows the probabilities of all the outcomes,” while typing in the commands:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> hs = np.arange(0, 40)
>>> fHs = rvH.pmf(hs)
```

code
2.1.20

```
>>> plt.stem(fhS)
```

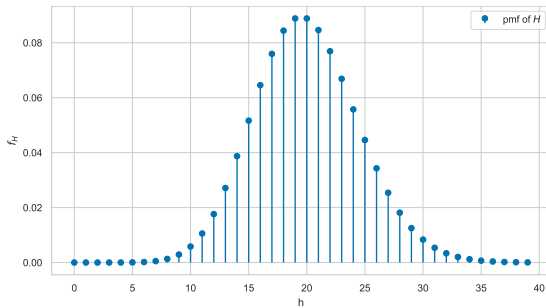


Figure 2.12: Plot of the probability mass function of the Poisson model with parameter $\lambda = 20$. The possible outcomes are clustered around $h = 20$ with most of the probability mass falling in the interval $[15, 25]$.

Desiring to keep the conversation going, you ask “Other questions?”

“I have one.” says Charlotte from software engineering. “I want to know, what is the maximum number of failures that I should plan for.”

“I can’t answer that question because, theoretically speaking, any number of failures can occur. What I can do is give you a 95% confidence interval,” you explain as you’re typing this in:

```
>>> rvH.ppf(0.95)
28
```

code
2.1.21

To explain what this number means, you say “95% of the data centres like ours will not observe more than 28 failures.” If you plan for 28 failures as the worst-case scenario, you know there is only a 5% chance that what happens next month will not be covered. Charlotte seems satisfied with the estimate.

David from the marketing department has a question. “Is this thing that you just did AI?” he asks, thinking about how he can use this in the webcopy for the new company website.

“I suppose you *could* say that, since we’re using probability distributions and probability distributions are also used in AI,” you explain, stretching the definitions.

“What about blockchain? Are we using a blockchain for this?”

“No blockchain,” you interject briskly. “Listen David, let’s proceed one buzzword at a time. You can have ‘AI’ for now. Show me you can sign \$1M worth of new clients using the ‘AI’ buzzword, then come back to me, and I’ll find another buzzword for you.”

“Okay, deal! I can work with that. AI is hot these days.”

Looking around the room, you sense the meeting is coming to a close. Everyone is feeling good about their first data science

experience. You decide to wrap things up with some random number generation. “To close the meeting, let me show you some examples of the possible numbers of failures we can expect to see during the next year,” you say while running the command needed to generate 12 random samples from the random variable rvH :

```
>>> rvH.rvs(12)
[20, 26, 18, 23, 13, 23, 22, 15, 26, 21, 19, 11]
```

code
2.1.22

You hear several low-level “wows” in the room as the concept of random variable finally sinks in. Simulations of real-world data always work! Finally people get it—the average is 20, but the number of failures can vary a lot around that average.

Later that day, you receive a followup email from Emily from the purchasing department. She wants an estimate of the total cost she should budget for replacement hard disks. The base price is \$200/disk, but it is reduced to \$150/disk if you buy 20 or more disks. In other words, Emily is asking you to compute $E_H[\text{cost}(H)]$, where the function $\text{cost}(h)$ describes the cost of purchasing h replacement disks. To compute the answer, you first define a Python function for the cost:

```
>>> def cost(h):
    if h >= 20:
        return 150*h
    else:
        return 200*h
```

code
2.1.23

You can then find the expected value of the function cost by computing the sum over all the possible outcomes, weighing the cost in each case by the probability of this outcome to occur.

```
>>> sum([cost(h)*rvH.pmf(h) for h in range(0,100+1)])
3381.42
```

code
2.1.24

Note we truncated the summation up to $h = 100$ because the probabilities $f_H(101)$, $f_H(102)$, etc. are negligibly small.

I hope reading about this real-world scenario convinced you of the general usefulness of the computer models defined in `scipy.stats` for doing probability calculations. The methods available on the random variable object `rvX` provide us with ways to compute all the quantities we introduced in this section using math equations. This means you don’t have to worry about memorizing all the math formulas, you just need to learn how to import one of the pre-defined probability models in `scipy.stats`, initialize the model with the correct parameters, then use its methods for the probability calculations you need. In the above example, we didn’t have to manually input the complicated-looking math formula $f_H(h) =$

$\frac{\lambda^h e^{-\lambda}}{h!}$ for the Poisson distribution, because the method `rvH.pmf(h)` already contains this formula!

To create a computer model for a random variable based on one of the models defined in `scipy.stats`, use the code `rvX = <model>(<params>)`, where `<model>` is the name of the model you imported from `scipy.stats`, and `<params>` is a comma-separated list of model parameters. The parameters will depend on the model. You'll learn about the most important discrete distribution in Section 2.3. See also the Appendix ?? for a complete list of distributions and their parameters.

Exercises

In these exercises, we'll explore the methods of random variable objects created from `scipy.stats` models. Exercises E2.14 and E2.15 will use the `randint(alpha, beta+1)` model, which corresponds to the discrete probability distribution $U_d(\alpha, \beta)$. In exercise E2.16, we'll use the Poisson model `poisson(lam) = Pois(λ)`.

You'll need to use a Jupyter notebook and run appropriate commands to import models from `scipy.stats`, create random variable objects, then call the methods on these objects to answer the questions.

E2.14 Let's reproduce the calculations related to the die-roll random variable D , which we introduced in Example 2.

- a) In a Jupyter notebook, import the `randint` (discrete uniform distribution) model from `scipy.stats`.
- b) Create the computer model for the random variable `rvD` by initializing using `rvD = randint(alpha, beta+1)` for an appropriate choice of start and stop parameters `alpha` and `beta`.
- c) Evaluate the probability mass function of the random variable D (`rvD.pmf`) for all inputs d in the range from 1 to 6. Do the values sum to one?
- d) Find value of the cumulative distribution function $F_D(4)$ by computing the sum of `rvD.pmf` values over the appropriate range of inputs.
- e) Compute $F_D(4)$ using the method `rvD.cdf`, and confirm it is the same as your previous answer.
- f) Compute the mean μ_D and the variance σ_D^2 of the random variable D by calling the appropriate methods on `rvD`.
- g) Compute the expected value $\mathbb{E}_D(w)$ of a game described in Example 4, with the winnings function is $w : \{1, 2, 3, 4, 5, 6\} \rightarrow \mathbb{R}$ defined in code 2.1.9.

E2.15 Consider the random variable D_{20} which describes the out-

come of rolling a 20-sided die. The random variable D_{20} follows the discrete uniform distribution $\mathcal{U}_d(1, 20)$, which assigns equal probability to all values in the sample space $\{1, 2, 3, \dots, 20\}$.

- a) Create the computer model for the random variable `rvD20` using the code `rvD20 = randint(alpha, beta+1)`, for an appropriate choice of start and stop parameters `alpha` and `beta`.
- b) Evaluate the probability mass function of the random variable D_{20} (`rvD20.pmf`) for the input $d = 7$.
- c) Compute $F_{D_{20}}(4)$ using the sum of `rvD20.pmf` values over the appropriate range of inputs.
- d) Compute $F_{D_{20}}(4)$ using the method `rvD20.cdf`.
- e) Compute the mean $\mu_{D_{20}}$, the variance $\sigma_{D_{20}}^2$, and the standard deviation $\sigma_{D_{20}}$ of the random variable D_{20} by calling the appropriate methods on `rvD20`.

E2.16 Computer memory errors can be modelled using a Poisson distribution. Consider the random variable M that describes the number of memory errors that will occur during a given time period. You know from past observations that the average number of memory errors is 40.

- a) In a Jupyter notebook, import the `poisson` model from `scipy.stats`.
- b) Create the random variable `rvM` from the `poisson` model to describe the number of memory errors by choosing the appropriate parameter λ when initializing the model.
- c) Compute the mean μ_M , the variance σ_M^2 , and the standard deviation σ_M of the random variable M by calling the appropriate methods on `rvM`.
- d) Compute $\Pr(\{33 \leq M \leq 44\})$ using the sum of `rvM.pmf` values over the appropriate range of inputs.
- e) Compute $\Pr(\{33 \leq M \leq 44\})$ using the method `rvM.cdf`.
- f) Find the smallest value $m_{0.95}$ such that $\Pr(\{M \leq m_{0.95}\}) \geq 0.95$.
- g) Generate 10 observations from the random variable `rvM`.

2.1.6 Discussion

The focus of this section was to define the basic notions of probability theory, like random variables and probability distributions. These are the bread-and-butter concepts of probabilistic reasoning. We'll now discuss some extra other topics that are not essential for understanding the basics of probability theory, but are still worth a brief mention.

Bulk of a distribution

We're often interested in calculating an interval that contains "the bulk" of the distribution f_X . We want to find an interval (a subset of the sample space) that contains most of the observations of the random variable X . If we want to compute the interval that contains 95% of the probability mass, we can use the values $F_X^{-1}(0.025)$ and $F_X^{-1}(0.975)$.

For example, the interval that contains 95% of the probability mass for the hard disk failures distribution f_H , is given by $I_{0.95} = [12, 29] = \{12 \leq H \leq 29\}$. We can verify this, by computing the probability of the outcome $H \in [12, 29]$, which gives us

$$\Pr(\{H \in I_{0.95}\}) = f_H(12) + f_H(13) + \cdots + f_H(29) = 0.9568.$$

If we make a prediction that $H \in I_{0.95}$, we'll be correct 95.68% of the time. See Figure 2.13 for an illustration of the probability mass contained in that interval.

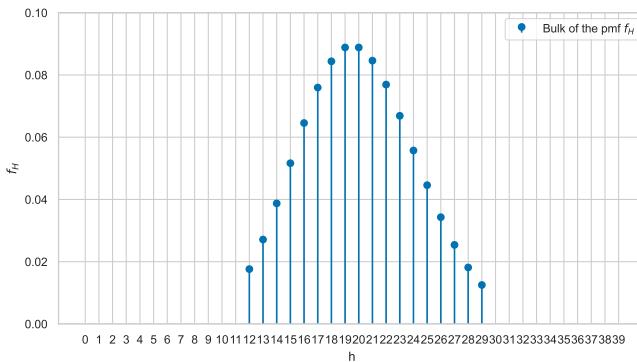


Figure 2.13: The probability mass of the function f_H over the interval $I_{0.95} = [12, 29]$ is 0.9568. Informally, we say this is "bulk" of the weight of the distribution of the random variable H lies. These are the most likely outcomes.

The technical term for the interval $I_{0.95}$ is a 95% *confidence interval*, meaning we're 95% confident that future outcomes of the random variable H will fall in this interval. This means, if we generate millions of observations from the random variable H , in the long run, 95% of these observations will be contained in the interval $[12, 29]$. Confidence intervals play an important role in statistics: we can provide a confidence interval whenever we estimate some quantity. We'll learn more about confidence intervals in Chapter ??.

Tails of a distribution

Conversely, the “tails” of the distribution contain the unlikely outcomes for the random variable. Figure 2.14 shows the tails of the distribution f_H , which are defined as the outcomes that are unlikely to occur. The probability of observing an outcome that is outside the interval $I_{0.95}$ is less than five percent:

$$\Pr(H \notin I_{0.95}) = \Pr(\{H < 12\}) + \Pr(\{H > 29\}) = 0.0432,$$

or 4.32%, which is a very unlikely event. The probability mass of the tails is shown in Figure 2.14. The probability in the tails is the complement of the probability mass shown in Figure 2.13.

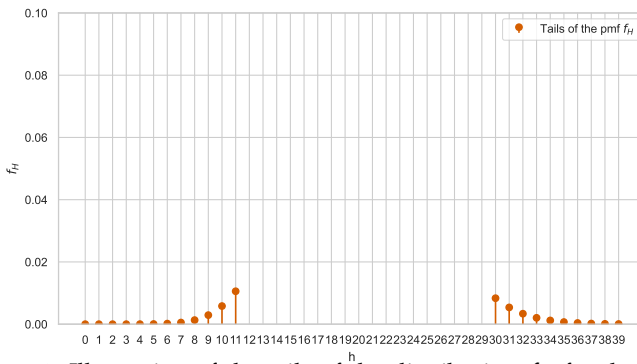


Figure 2.14: Illustration of the tails of the distribution f_H for the random variable H . The probability mass shown corresponds to the outcomes that are outside the interval $I_{0.95} = [12, 29]$. The total probability mass in the tails is 0.0432, or 4.32%, which we qualify as “very unlikely.”

Observations in the tails of the distribution are deemed “surprising” and “unexpected.” The notion of “unexpected outcome” plays a central role in the statistical concept of hypothesis testing (Section 3.X), which is another of one of the main ideas we’ll study in Chapter ??.

Interpretations of probability theory

We talked a lot about probability calculations, but we never explained what probabilities mean. It turns out there are two very distinct schools of thought on this matter: the *frequentist* and the *Bayesian* interpretations of probability theory.

Frequentist interpretation One conceptual interpretation of probability theory is to define probabilities as relative frequencies of occurrence. In the frequentist paradigm, the probability mass function

f_C of the coin toss random variable C describes how likely we are to observe heads and tails if we were to toss the coin thousands or millions of times: (c_1, c_2, \dots, c_n) , where n is 1 000, 1 000 000, or more. The probability of the outcome heads is defined as the relative frequency of heads outcomes, if we were to draw an infinite sequence of observations:

$$f_C(\text{heads}) \stackrel{\text{def}}{=} \frac{\text{count of } c_i = \text{heads}}{n} \text{ as } n \text{ goes to infinity.}$$

The probability of $f_C(\text{tails})$ is analogously defined as the relative frequency of the outcome $c_i = \text{tails}$, as n goes to infinity.

The frequentist interpretation of probability theory is well suited for describing random outcomes governed by a well-defined rule or theory. For example, if we assume a coin is fair, then the probabilities of heads and tails are equal: $f_C(\text{heads}) = f_C(\text{tails}) = \frac{1}{2}$.

Bayesian interpretation Another conceptual interpretation of probability theory is to consider probability distributions as representations of our state of *knowledge* or *beliefs* about a given phenomenon. According to the Bayesian point of view, the probability distribution f_C describes our “best guess” about the distribution of the random variable C , based on our current knowledge.

This approach to probability theory is named after Thomas Bayes, an 18th century mathematician who came up with a general principle for updating our beliefs about f_C based on data observations. For example, we start out with some initial belief about the possible distributions of the coin before we observe any coin tosses. We then toss the coin n times to obtain the sequence of observations (c_1, c_2, \dots, c_n) , each c_i being either heads or tails. *Bayes’ rule* tells us how to update our beliefs about the possible distributions after observing the data. We’ll describe Bayes’ rule in Section 2.2.4, after we introduce the notions of conditional and marginal probability distributions. In Chapter ??, we’ll learn more about the applications of Bayes’ rule in statistics (see Section ??).

The choice of interpretation for probability doesn’t make a big difference for practical probability calculations—both interpretations use the same tools, probability distributions, random variables, and their observations. Later on in the statistics chapter, we’ll see that frequentist and Bayesian interpretation lead to different types of guarantees that we can give when we report statistics results.

Next steps

Okay we're done. Congratulations on getting through all the definitions, formulas, and calculations introduced in this section. I know it was a lot of stuff, but the good news is that you've now seen all the key building blocks. If you made it this far, you can probably handle all the rest of the chapter too!

Let me give you a little preview of what is coming up. In the next section, we'll talk about situations that involve multiple random variables. Instead of a single random variable X described by a probability distribution function f_X , we'll talk about pairs of random variables (X, Y) with a *joint* probability distribution function f_{XY} . The analysis of multiple random variables is very important in statistics, so we'll make sure to set up a solid foundation for it in Section 2.2, and again in Section 2.8 when we'll study the properties of sequences of n random variables (X_1, X_2, \dots, X_n) that come from the same distribution.

In the second half of the chapter, we'll study continuous random variables, which are used for describing smoothly varying quantities like lengths, weights, time intervals, etc. Doing probability calculations using continuous random variables requires integration, which is a calculus procedure for calculating the total amount of quantities that change over time. Don't worry if you haven't studied calculus before: we'll present a self-contained introduction to calculus concepts in Section 2.4. We'll then give the precise definitions and formulas for continuous random variables in Section 2.5.

Throughout the chapter, we'll show numerous examples of computations based on the computer models defined in `scipy.stats`. By the end of the chapter you'll have gained experience with all important discrete probability models like `randint`, `poisson`, `bernoulli`, `binom`, etc. and all the continuous models like `uniform`, `norm`, `expon`, `t`, `chi2`, etc. See Appendix ?? for a complete list of probability distributions and their parameters.

Knowing these essential building blocks of probability theory will prepare you for the study of statistics chapters ?? and ??.

Links

[Beautiful visualizations of probability theory topics]

<https://seeing-theory.brown.edu/basic-probability/>

[Some relevant pages from Wikipedia]

https://en.wikipedia.org/wiki/Probability_theory

https://en.wikipedia.org/wiki/Probability_distribution

[Comprehensive list of hundreds of probability distributions]

https://en.wikipedia.org/wiki/List_of_probability_distributions

[Probability theory chapter from Simon Hubbert's book]

https://bookdown.org/S_hubbert/mathematics_of_financial_derivatives/Prob-Th.html

[Excellent tutorial and examples of probability mass functions]

<https://www.probabilitycourse.com/chapter3/313pmf.php>

2.2 Multiple random variables

We're often interested in analyzing the relationship between two random variables, like X and Y . Instead of describing the unknown quantities X and Y separately, we can describe the joint variability of the pair (X, Y) using the *joint probability distribution* f_{XY} . Different types of joint probability distributions f_{XY} can be used to model different types of relationships between the two random variables.

In this section, we'll go over some basic definitions and formulas used for calculations with multiple random variables. We'll also introduce some new math concepts like the *conditional probability distribution* $f_{Y|X}$, read "probability of Y given X ," which describes the uncertainty about Y that remains once we observe the value of X . The joint probability distribution f_{XY} and the conditionals $f_{Y|X}$ and $f_{X|Y}$, and useful "tools" for modelling the relations between random variables, as you'll see through the examples in this section.

Unlike the previous section which was very dense in new concepts, math formulas, and code examples, this section is comparatively light on new material and won't demand too much cognitive effort. You deserve a break after all the math formulas we went through in the previous section. The formulas and calculations in this section are simple extensions of what we learned about random variables in the previous section. Since we're dealing with multiple variables, the graphical representation of probability distributions will look different, but apart from that, there won't be any new ideas.

2.2.1 Definitions

Consider the pair of random variables (X, Y) defined over the sample space $\mathcal{X} \times \mathcal{Y}$, which is the *product* of the sample spaces for the two random variables \mathcal{X} and \mathcal{Y} . We call $\mathcal{X} \times \mathcal{Y}$ the *joint sample space* of the random variables (X, Y) . A particular outcome in the joint sample space looks like a pair of numbers (x, y) , where x is the value observed for the random variable X , and y is the value observed for the random variable Y . Geometrically speaking, the joint sample space is a two-dimensional region.

At this point, it might be worth clarifying again that a "joint sample space" is not some kind of establishment that you visit to try different types of marijuana smokables. Rest assured, learning about multi-variable probability distributions will lead you to some all-natural experiences of *knowledge buzz*, as you learn about some of the most important probability ideas, which form the foundation of statistics.

- X, Y : a pair of random variables

- $\mathcal{X} \times \mathcal{Y}$: the *joint sample space* of the random variables X and Y
- $f_{XY}(x, y)$ the *joint probability mass function* of the random variables X and Y . The function f_{XY} has the form $f_{XY}: \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$, and it tells us the probability of each of the possible outcomes:

$$f_{XY}(x, y) \stackrel{\text{def}}{=} \Pr(\{X = x, Y = y\}),$$

for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.

The math machinery for working with multiple random variables is similar to what we saw in the previous section for the single-variable case, but we'll be working with functions like $f_{XY}(x, y)$ that depend on two or more inputs, instead of single-variable functions like $f_X(x)$.

The outcomes in the joint sample space consists of subsets of the sample space. For example, the outcome where X takes on a value between $x = a$ and $x = b$, and Y takes on a value between $y = c$ and $y = d$ is written as $A = \{a \leq X \leq b, c \leq Y \leq d\}$ in set notation, or alternatively as $A = \{(X, Y) \in \mathbb{N} \times \mathbb{N} \mid X \in [a, b], Y \in [c, d]\}$. Another, more compact, representation of this outcome is as a product of intervals $A = [a, b] \times [c, d]$. Geometrically speaking, this outcome describes a rectangular region with width $b - a$ and height $d - c$. The probability of this outcome A is obtained by calculating the sum of the probability mass function f_{XY} over all the values in the subset A :

$$\Pr(A) = \sum_{(x, y) \in A} f_{XY}(x, y) = \sum_{x=a}^{x=b} \sum_{y=c}^{y=d} f_{XY}(x, y).$$

Up until here there is nothing new going on. The concept of a joint probability mass function f_{XY} is directly analogous to the probability mass functions we saw in the previous section, but with more dimensions. You have to trust me on this one—double summations formulas might look intimidating, but there is nothing fancy going on. It's still the same idea of calculating the “total” amount of f_{XY} over a set of outcomes.

Marginal and conditional distributions

Starting from the joint probability mass function $f_{XY}(x, y)$, we can define the *marginal* and *conditional* probability distributions.

The process of marginalization describes the uncertainty in one random variable when we don't know the other random variable:

- $f_X(x) \stackrel{\text{def}}{=} \sum_{y \in \mathcal{Y}} f_{XY}(x, y)$ is the *marginal distribution* for the random variable X . The marginal distribution f_X describes

the uncertainty in the random variable X when the random variable Y is unknown.

- $f_Y(y) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} f_{XY}(x, y) dx$ is the *marginal distribution* for the random variable Y .

When a random variable is unknown, we model our ignorance of its value by summing over all the possible values it can take.

Conditional distributions

The process of conditioning describes what happens when we know one of the random variables, and we want to quantify the uncertainty that remains in the other random variable:

- $f_{X|Y}(x|y) \stackrel{\text{def}}{=} \Pr(\{X = x\}|\{Y = y\})$ is the conditional distribution of X given Y . The vertical bar “|” is read “given.” Suppose we know the outcome $\{Y = y\}$ has occurred. What is the distribution of the random variable X given that $\{Y = y\}$ has been observed? The conditional distribution function $f_{X|Y}(x|y)$ tells us the answer to this question.
- $f_{Y|X}(y|x) \stackrel{\text{def}}{=} \Pr(\{Y = y\}|\{X = x\})$ is the conditional distribution of Y given X .

Conditional distributions $f_{X|Y}$ and $f_{Y|X}$ are the main tools we use to model relations between random variables, as you see in the rest of this section.

2.2.2 Joint probability distributions

The joint probability distribution f_{XY} is our main tool for modelling relationships between two random variables X and Y . By choosing the appropriate function f_{XY} , we can describe and model various relationships between random variables. Since the probability mass function $f_{XY}(x, y)$ has two inputs, we can plot the distribution as a two-dimensional *contour plot*, with darker shaded regions indicating higher values, as shown in Figure 2.15. The exact formula for the probability mass function $f_{XY}(x, y)$ shown in Figure 2.15 is not important; we’re just using this probability mass function as an example.

Like all probability distributions, the joint probability distribution has nonnegative values, $f_{XY}(x, y) \geq 0$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$, and the total amount of probability is one $\sum_{(x, y) \in \mathcal{X} \times \mathcal{Y}} f_{XY}(x, y) = 1$.

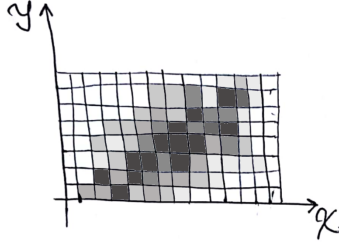


Figure 2.15: Graphical representation of a joint probability distribution $f_{XY} : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$, where $|\mathcal{X}| = 14$ and $|\mathcal{Y}| = 8$. The darkness of each square (x, y) represents is proportional to its mass.

Example 1: two coin tosses Consider the random experiment in which we toss a fair coin two times. We'll model the outcome of each coin toss as a random variable $C \in \{\text{heads}, \text{tails}\}$, with probability mass function f_C defined as $f_C(\text{heads}) = \frac{1}{2}$ and $f_C(\text{tails}) = \frac{1}{2}$.

The joint sample space for the two coin tosses contains four possible outcomes:

$$\{\text{heads}, \text{tails}\} \times \{\text{heads}, \text{tails}\} = \{(\text{heads}, \text{heads}), (\text{heads}, \text{tails}), (\text{tails}, \text{heads}), (\text{tails}, \text{tails})\}.$$

The joint probability mass function $f_{C_1 C_2}(c_1, c_2)$ that describes the two coin toss experiment is

$$f_{C_1 C_2}(c_1, c_2) = f_C(c_1) \cdot f_C(c_2) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

Note the joint probability mass function $f_{C_1 C_2}$ is the product of the probability mass functions for the individual coins f_C . Figure 2.16 shows a stem plot of the joint probability mass function $f_{C_1 C_2}$. The height of each stem corresponds to the probability of this outcome.

Figure 2.17 shows a graphical representation of $f_{C_1 C_2}$ as a tree diagram, which describes the sequence of coin tosses. Reading the diagram from left to right, we coin toss C_1 causes a “split” in the world of possibilities into two branches, then the second coin C_2 causes another split, so in the end there are four possible outcomes. To compute the probability of each outcome, we work our way backward and multiply together the probabilities observed along each branch.

The total probability over all possible outcomes satisfies Kolmogorov's second axiom:

$$\sum_{c_1 \in \{\text{heads}, \text{tails}\}} \sum_{c_2 \in \{\text{heads}, \text{tails}\}} f_{C_1 C_2}(c_1, c_2) = 1.$$

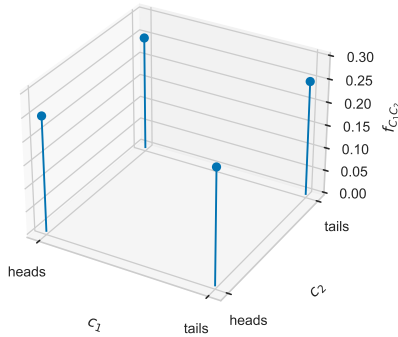


Figure 2.16: Graph of the joint probability mass function $f_{C_1 C_2}$ for the two coin tosses.

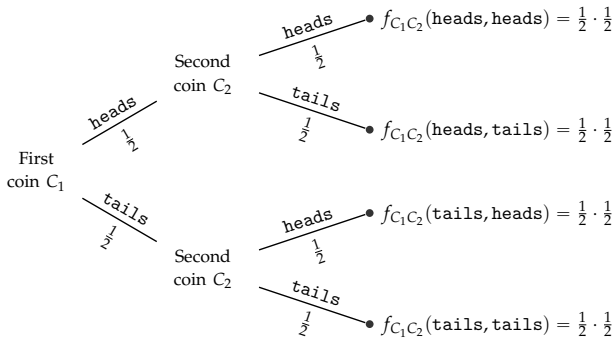


Figure 2.17: Tree diagram of the joint probability mass function $f_{C_1 C_2}$ for two coin tosses.

If we want to compute the probability of “one heads,” we need to sum together all the possible outcomes that contain one heads, of which there are two:

$$\begin{aligned} \Pr(\{\text{one heads}\}) &= f_{C_1 C_2}(\text{heads}, \text{tails}) + f_{C_1 C_2}(\text{tails}, \text{heads}) \\ &= \frac{1}{4} + \frac{1}{4} = \frac{1}{2}. \end{aligned}$$

Example 2: rolling a pair of dice Consider the pair of random variables (D_1, D_2) that represent the outcome of rolling two balanced dice. The random variables D_1 and D_2 are both described by probability mass function $f_D(d) = \frac{1}{6}$, for all d in the sample space $\{1, 2, 3, 4, 5, 6\}$. Their joint probability mass function $f_{D_1 D_2}$ is the product of the two copies of the distribution f_D :

$$f_{D_1 D_2}(d_1, d_2) = f_D(d_1) \cdot f_D(d_2) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36}.$$

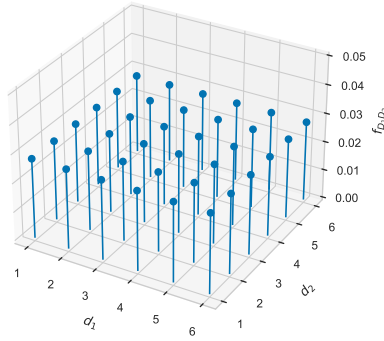


Figure 2.18: Graph of the joint probability mass function $f_{D_1 D_2}$ for the two die rolls.

Suppose we want to calculate the probability of “rolling a seven,” which is described by the equation $D_1 + D_2 = 7$, which corresponds to the set of outcomes $\{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}$. The probability of this outcome is the sum of $f_{D_1 D_2}$ over this set of outcomes:

$$\begin{aligned}
 \Pr(\{D_1 + D_2 = 7\}) &= \Pr(\{(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)\}) \\
 &= f_{D_1 D_2}(1, 6) + f_{D_1 D_2}(2, 5) + f_{D_1 D_2}(3, 4) \\
 &\quad + f_{D_1 D_2}(4, 3) + f_{D_1 D_2}(5, 2) + f_{D_1 D_2}(6, 1) \\
 &= 6\left(\frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6} + \frac{1}{6}\right) = \frac{1}{6} = 0.1\bar{6}.
 \end{aligned}$$

Geometrically speaking, this summation corresponds to adding up the total length of stems on the diagonal line that goes from $(1, 6)$ until $(6, 1)$, as shown in Figure 2.19.

Marginal distribution functions

The *marginal probability mass function* f_X is obtained from the joint distribution f_{XY} by summing over all possible outcomes of the variable Y :

$$f_X(x) \stackrel{\text{def}}{=} \sum_{y \in \mathcal{Y}} f_{XY}(x, y).$$

The idea for a marginal distribution f_X is to get rid of the Y randomness, which corresponds to a description of the random variable X when the random variable Y is unknown. The name *marginal distribution* comes from the procedure we use to compute it, by summing all y values for a given x and writing the total in the margin. See Figure 2.20 for an illustration, and also recall Table ??

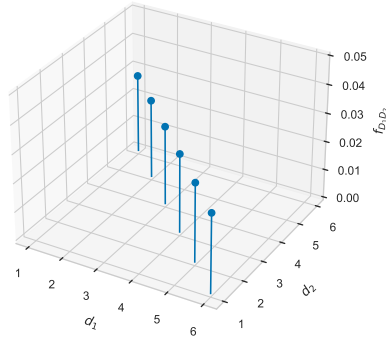


Figure 2.19: Subset of the weights of the joint probability mass function $f_{D_1 D_2}$ that corresponds to the outcome $D_1 + D_2 = 7$.

on page ??, where we used a similar procedure for computing the marginal frequencies in a dataset using a two-way table.

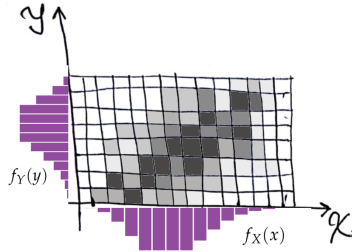


Figure 2.20: Marginal distribution f_X is obtained by summing all the values of f_{XY} in each column. Marginal distribution f_Y is obtained by summing all the values of f_{XY} in each row.

The marginal distributions f_Y is obtained from the joint distribution f_{XY} similarly by summing over all possible values of the variable X :

$$f_Y(y) = \sum_{x \in \mathcal{X}} f_{XY}(x, y).$$

The marginal distribution f_Y describes the randomness of Y when we don't know the value of X .

The marginal distributions of the first coin toss f_{C_1} is equal to the probability of a single coin toss f_C , as shown in the following

calculation:

$$\begin{aligned}
 f_{C_1}(c_1) &= \sum_{c_2 \in \{\text{heads}, \text{tails}\}} f_{C_1 C_2}(c_1, c_2) \\
 &= \sum_{c_2 \in \{\text{heads}, \text{tails}\}} f_C(c_1) f_C(c_2) \\
 &= f_C(c_1) \sum_{c_2 \in \{\text{heads}, \text{tails}\}} f_C(c_2) \xrightarrow{1} \\
 &= f_C(c_1).
 \end{aligned}$$

Similarly, the marginal of C_2 is $f_{C_2} = f_C$.

The marginal distributions f_{D_1} and f_{D_2} of the two-die rolls joint distribution $f_{D_1 D_2}$ correspond to the probabilities of a single die roll:

$$f_{D_1} = f_D \quad \text{and} \quad f_{D_2} = f_D.$$

Indeed, whenever the joint probability distribution for two random variables is the product of two distributions $f_{XY} = f_X \cdot f_Y$, the marginals of this joint distribution will be f_X and f_Y . The technical term for this “product structure” of joint probability mass functions is *independence*, meaning the probabilities of the X outcomes are independent (not related to) the outcomes of the random variable Y .

The opposite of independent random variables, are “dependent” random variables, meaning the outcome of one variable depends on the outcome of another. In the next section, we’ll learn how to describe the dependence between random variables using *conditional probability distributions*.

2.2.3 Conditional probability distributions

The *conditional probability mass* functions $f_{X|Y}$ and $f_{Y|X}$ are defined as follows:

$$f_{X|Y}(x|y) \stackrel{\text{def}}{=} \frac{f_{XY}(x, y)}{f_Y(y)} \quad \text{and} \quad f_{Y|X}(y|x) \stackrel{\text{def}}{=} \frac{f_{XY}(x, y)}{f_X(x)}.$$

Intuitively, the conditional distribution $f_{X|Y}(x|y)$ has the effect of “focussing” the sample space on the subset of outcomes where $\{Y = y\}$. We divide by the normalization factor $f_Y(y)$ in order to make $f_{X|Y}$ a valid distribution (sums to one) over the sample space \mathcal{X} . See Figure 2.21 for an illustration.

The vertical bar is pronounced “given” and describes situations where the realization of some random variables is known. For example, the conditional distribution $f_{Y|X}(y|x_a)$ describes the probabilities

of the random variable Y , given we know the value of the random variable X is x_a . The distribution $f_{Y|X}(y|x_b)$ describes the separate case when $X = x_b$, and in general, there is a different distribution for each of the possible $x \in \mathcal{X}$.

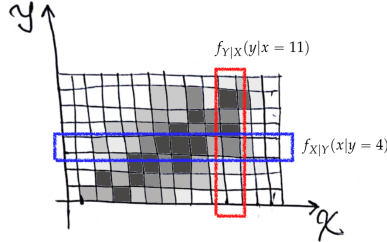


Figure 2.21: Conditional distributions $f_{Y|X}(y|x)$ represent different vertical slices through the joint distribution. Similarly, conditional distributions $f_{X|Y}(x|y)$ are horizontal slices of the joint distribution.

If you know $f_{Y|X}$ and f_X , then you can reconstruct the joint distribution f_{XY} by computing:

$$f_{XY}(x, y) = f_{Y|X}(y|x) \cdot f_X(x).$$

This is called the *chain rule* of probability theory. Convince yourself that the sequential description of outcomes shown on the right side of the equation is a valid description of the randomness in the random variables (X, Y) .

We can use a tree diagram to provide a visual explanation of the chain rule $f_{XY}(x, y) = f_{Y|X}(y|x) \cdot f_X(x)$. Suppose $\mathcal{X} = \{a, b\}$ and $\mathcal{Y} = \{1, 2\}$, which means the joint sample space is four possible outcomes $\{a, b\} \times \{1, 2\} = \{(a, 1), (a, 2), (b, 1), (b, 2)\}$. The tree diagram in Figure 2.22 illustrates the sequence of steps “observe X then observe Y .”

Let’s look at the outcome $(X, Y) = (b, 1)$ in particular. The probability of this outcome can be obtained by multiplying together the probability of the outcomes $\{X = b\}$ and the outcome $\{Y = 1|X = b\}$: $f_{XY}(b, 1) = f_X(b)f_{Y|X}(1|b)$, where $f_X(b)$ and $f_{Y|X}(1|b)$ are the two probability weights on the “path” to this outcome. The probabilities of all outcomes are obtained similarly, by multiplying the probabilities along the paths that lead to them.

The chain rule can also be applied in the opposite order, by imagining the “observe Y then observe X ” scenario. If we know the distributions f_Y and $f_{X|Y}$, we can write the joint distribution f_{XY} as $f_{XY}(x, y) = f_{X|Y}(x|y) \cdot f_Y(y)$.

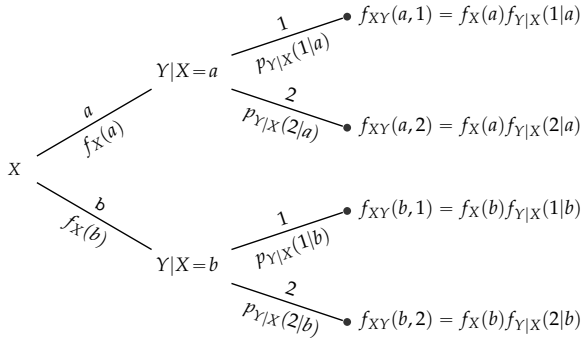


Figure 2.22: Tree diagram of the sequential computations of the probabilities f_{XY} based on the chain rule.

The following examples illustrate joint probability mass functions f_{XY} defined in terms of a marginal distribution f_X and the conditional distributions $f_{Y|X}$.

Example 3: coin toss followed by a conditional die roll Suppose you're playing a game that involves a coin toss followed by rolling one of two dice. Let C denote the outcome of the coin toss and D denote the outcome of the die roll. First you toss a fair coin, described the random variable $C \in \{\text{heads}, \text{tails}\}$. The outcome of the coin toss determines which die to roll. If the coin comes out heads you'll throw the six sided die, else if the coin comes out tails you'll throw the tetrahedral die.

The probability distribution of the coin is $p_X(\text{heads}) = p_X(\text{tails}) = \frac{1}{2}$. The probability mass function of the six-sided die is $p_{Y|X}(y|\text{heads}) = \frac{1}{6}$, for $y \in \{1, 2, 3, 4, 5, 6\}$. The probability distribution for the tetrahedral (four sides) die is $p_{Y|X}(y|\text{tails}) = \frac{1}{4}$, for $y \in \{1, 2, 3, 4\}$.

Since the sample space \mathcal{Y} has only six possible outcomes, we can express the conditional probability mass functions by writing them as lists of six values:

$$p_{Y|X}(y|\text{heads}) = [\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}], \quad p_{Y|X}(y|\text{tails}) = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, 0].$$

The marginal probability distribution $p_Y(y)$ can then be written as

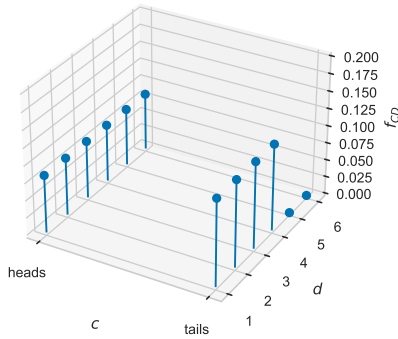


Figure 2.23: Graph of the joint probability mass function f_{CD} for a game involving a coin toss, and rolling a six-sided die D or a tetrahedral die D_4 .

the weighted sum of the two conditional distributions:

$$\begin{aligned}
 p_Y(y) &= p_{Y|X}(y|\text{heads})p_X(\text{heads}) + p_{Y|X}(y|\text{tails})p_X(\text{tails}) \\
 &= \left[\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}\right] \cdot \frac{1}{2} + \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, 0\right] \cdot \frac{1}{2} \\
 &= \left[\frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}, \frac{1}{12}\right] + \left[\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, 0, 0\right] \\
 &= \left[\frac{5}{24}, \frac{5}{24}, \frac{5}{24}, \frac{5}{24}, \frac{1}{12}, \frac{1}{12}\right].
 \end{aligned}$$

Note the outcomes $\{1, 2, 3, 4\}$ for the random variable Y in the marginal distribution f_Y are more likely, since they can be observed either from the six-sided die or from the four-sided die.

We can verify that the sum of the marginal's weights satisfies Kolmogorov's second axiom: $\sum_{y=1}^{y=6} p_Y(y) = 4 \cdot \frac{5}{24} + 2 \cdot \frac{1}{12} = 1$.

Example 4: medical diagnostic test Consider a doctor interpreting the results of a diagnostic test that is meant to detect if a patient has a given virus. We can describe the “has virus” unknown as a random variable $V \in \{0, 1\}$, and the outcome of the test as a random variable $T \in \{1, 0\}$ (either positive or negative).

The doctor knows the current prevalence of the virus in the population is 3%, meaning $\Pr(V = 1) = 0.03$ for individuals randomly selected from the local population.

The sensitivity of the test is 90%, meaning if the probability of correctly detecting the virus in a patient who has the virus is 0.9. Expressed as a conditional distribution, this tells us, when $V = 1$, the conditional probability of the random variable T is

$$f_{T|V}(T = 1|V = 1) = 0.9 \quad \text{and} \quad f_{T|V}(T = 0|V = 1) = 0.1.$$

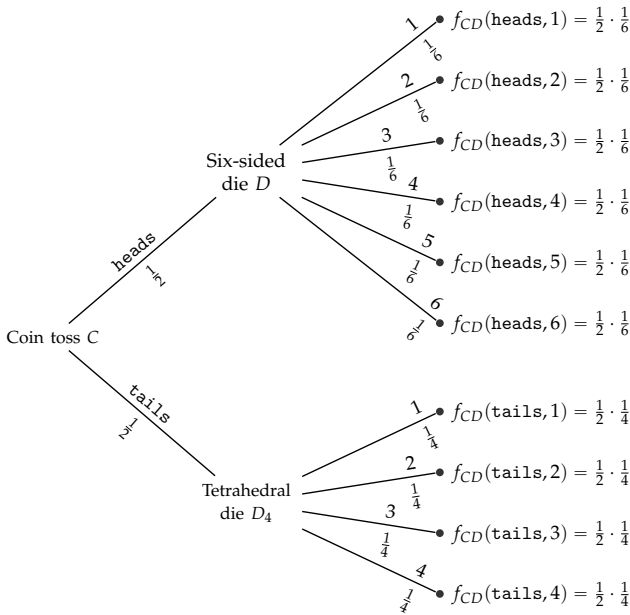


Figure 2.24: Tree diagram of the joint probability mass function f_{CD} for the game.

Note sensitivity of 90% means this test will fail to detect the virus in 10% of patients that have it. Another way to describe this situation is to say the test has a *false negative* rate of 10%.

The doctor also knows that the *specificity* of the test is 80%, meaning it correctly predicts the absence of the virus for patients that don't have it with probability 0.8:

$$f_{T|V}(T = 0|V = 0) = 0.8.$$

Equivalently, we can say *specificity* of 80% corresponds to a false positive rate of 20%, which is the probability of obtaining a positive test for a patient that doesn't have the virus:

$$f_{T|V}(T = 1|V = 0) = 0.2.$$

By combining the information for the general virus prevalence $\Pr(V = 1) = 0.03$, $\Pr(V = 0) = 0.97$, and properties of the test (sensitivity $f_{T|V=1}$ and specificity $f_{T|V=0}$), the doctor can obtain a complete view of the joint probability mass function f_{VT} for this population:

$$f_{VT}(v, t) = f_{T|V}(t|v)f_V(v).$$

The values of the probability mass function are

$$f_{VT}(0,0) = f_{T|V}(0|0)f_V(0) = 0.8 \cdot 0.97 = 0.776,$$

$$f_{VT}(0,1) = f_{T|V}(1|0)f_V(0) = 0.2 \cdot 0.97 = 0.194,$$

$$f_{VT}(1,0) = f_{T|V}(0|1)f_V(1) = 0.1 \cdot 0.03 = 0.003,$$

$$f_{VT}(1,1) = f_{T|V}(1|1)f_V(1) = 0.9 \cdot 0.03 = 0.027.$$

Figure 2.25 shows a graph of the joint probability mass function f_{VT} .

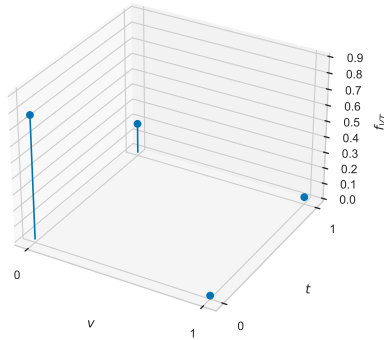


Figure 2.25: Graph of the joint probability mass function f_{VT} .

We can also represent the joint distribution f_{VT} by constructing a tree diagram, as shown in Figure 2.26.

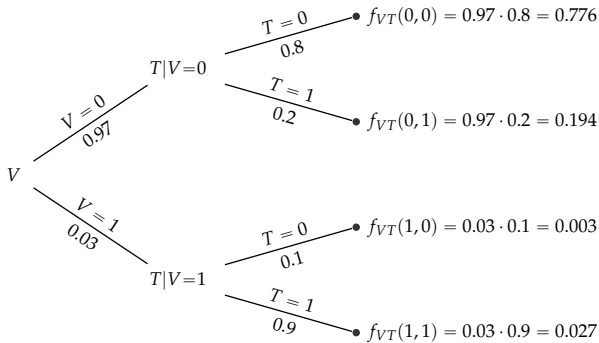


Figure 2.26: Tree diagram of the joint probability mass function f_{VT} .

The marginal distribution of the test outcomes is given by

$$f_T(t) = \sum_{v=0}^{v=1} f_{VT}(v,t) = f_{VT}(0,t) + f_{VT}(1,t),$$

The probability of observing a positive test is given by

$$\begin{aligned} f_T(1) &= f_{VT}(0, 1) + f_{VT}(1, 1) \\ &= f_{T|V}(1|0)p_V(0) + f_{T|V}(1|1)p_V(1) \\ &= 0.2 \cdot 0.97 + 0.9 \cdot 0.03 = 0.221. \end{aligned}$$

Note there are two contributions to this probability, either the patient has the virus and we correctly detected it (the second term $f_{T|V}(1|1)p_V(1)$), or the patient doesn't have the virus and the test result is a false positive (the first term $f_{T|V}(1|0)p_V(0)$).

The probability of observing a negative test is given by

$$\begin{aligned} f_T(0) &= f_{VT}(0, 0) + f_{VT}(1, 0) \\ &= f_{T|V}(0|0)p_V(0) + f_{T|V}(0|1)p_V(1) \\ &= 0.8 \cdot 0.97 + 0.1 \cdot 0.03 = 0.779. \end{aligned}$$

Again, the probability has two contributions: a contribution from the correct false test outcome for patients that don't have the virus, and a contribution from the false negative rate of the test for patients that have the virus.

Knowing the joint distribution f_{VT} and the marginal distribution f_T will allow the doctor to make accurate probability statements when interpreting the test result. In particular, the doctor can quantify the probability of having the virus if the result of the test is positive, $f_{V|T}(v|T = 1)$. We'll complete this calculation later in this section, after learning about the "math tool" we need to obtain the answer (Bayes' rule).

2.2.4 Probability formulas and rules

Based on the above definitions, we can derive several useful formulas that will help us when doing calculations with multiple random variables.

Chain rule

The chain rule states that we can decompose the joint distribution f_{XY} as the product of the conditional distribution $f_{Y|X}$ and marginal distribution f_X :

$$f_{XY}(x, y) = f_{Y|X}(y|x) \cdot f_X(x).$$

In words, this means the joint-uncertainty in (X, Y) can be broken down into X -uncertainty, and Y -given- X -uncertainty. Note this

formula is just a consequence of the way we defined conditional probability distribution $f_{Y|X}(y|x) \stackrel{\text{def}}{=} \frac{f_{XY}(x,y)}{f_X(x)}$.

We can extend the chain rule to three random variables X, Y, Z , to obtain the following decomposition:

$$f_{XYZ}(x, y, z) = f_{Z|XY}(z|x, y) \cdot f_{Y|X}(y|x) \cdot f_X(x).$$

This randomness in the joint distribution f_{XYZ} can be understood composed of three parts: X randomness, Y given X randomness, and Z given X and Y randomness. This is where the name “chain rule” comes from: it allows us to write the joint distribution f_{XYZ} as a chain of conditional distributions.

Law of total probability

Consider the random variables X and Y with sample spaces \mathcal{X} and \mathcal{Y} . The marginal distribution f_X is defined as the following summation:

$$f_X(x) = \sum_{y \in \mathcal{Y}} f_{XY}(x, y)$$

In words, this means the probability of the event $\{X = x\}$ can be obtained as the sum of joint probabilities over all possible outcomes of the random variable Y . If we define $\mathcal{Y} = \{y_1, y_2, \dots, y_n\}$, we can write the formula explicitly as

$$f_X(x) = f_{XY}(x, y_1) + f_{XY}(x, y_2) + \dots + f_{XY}(x, y_n).$$

The name total probability comes from the idea that we’re computing $f_X(x)$ by adding up the total of $f_X(x, y)$ for all possible outcomes of the random variable Y .

Bayes’ rule

Bayes’ rule is a simple observation that combines the chain rule and the law of total probability to obtain an extremely useful formula:

$$f_{Y|X}(y|x) = \frac{f_{XY}(x, y)}{f_X(x)} = \frac{f_{X|Y}(x|y)f_Y(y)}{f_X(x)} = \frac{f_{X|Y}(x|y)f_Y(y)}{\sum_{y' \in \mathcal{Y}} f_{X|Y}(x|y')f_Y(y')}.$$

The first equality follows from the definition of conditional probability distribution $f_{Y|X}(y|x) \stackrel{\text{def}}{=} \frac{f_{XY}(x,y)}{f_X(x)}$. The second equality follows from the chain rule $f_{XY}(x, y) = f_{Y|X}(y|x)f_X(x)$. Finally, the third equality is obtained by rewriting the denominator using the law of total probability: $f_X(x) = \sum_{y' \in \mathcal{Y}} f_{XY}(x, y')$, but using the different

summation variable y' to avoid confusion with the variable y used in the numerator.

Bayes' rule is a recipe for "inverting" the conditioning relation between two random variables: it allows us to write $f_{Y|X}$ in terms of an expression that involves $f_{X|Y}$ and f_Y only. This turns out to be a very useful procedure, since many real-world situations can be described by conditional distributions $f_{X|Y}$, but the questions we're interested in answering are related to $f_{Y|X}$.

In Section ?? we'll learn *Bayesian statistics*, which is an approach to statistical modelling and inference that is based on Bayes rule. For now, we'll just look at a simple example to illustrate the "inverting" conditional probabilities.

Example 4 continued The doctor wants to calculate the probability $f_{V|T}(v|t = 1)$, which describes the probability of a patient having the virus if the diagnostic test result is positive. The doctor knows the specificity and the sensitivity of the test, which describe the conditional distribution $f_{T|V}$, but we're interested in the distribution $f_{V|T}$. This seems like the perfect moment to apply Bayes' rule.

Let's start by writing the general formula for Bayes' rule, as it applies to the random variables V and T :

$$f_{V|T}(v|t) = \frac{f_{T|V}(t|v)f_V(v)}{\sum_{v' \in \{0,1\}} f_{T|V}(t|v')f_V(v')} = \frac{f_{T|V}(t|v)f_V(v)}{f_{T|V}(t|0)f_V(0) + f_{T|V}(t|1)f_V(1)}.$$

The second equation was obtained by expanding the summation, over the sample space for the random variable V , which consists of only two elements.

We're interested in the case when the patient has tested positive $t = 1$, so we can rewrite the general formula for this specific case:

$$f_{V|T}(v|t = 1) = \frac{f_{T|V}(1|v)f_V(v)}{f_{T|V}(1|0)f_V(0) + f_{T|V}(1|1)f_V(1)} = \frac{f_{T|V}(1|v)f_V(v)}{f_T(1)}.$$

Note we simplified the denominator, identifying the sum $f_{T|V}(1|0)f_V(0) + f_{T|V}(1|1)f_V(1)$ as the value of the marginal $f_T(1)$.

We can now compute the probability the patient has a virus $f_{V|T}(v = 1|t = 1)$, by substituting the relevant quantities:

$$f_{V|T}(1|t = 1) = \frac{f_{T|V}(1|1)f_V(1)}{f_T(1)} = \frac{0.9 \cdot 0.03}{0.221} = 0.122,$$

where we used the value $f_T(1) = 0.221$ we computed earlier (see Example 4).

Bayes' rule tells us the probability a patient has the virus, given they tested positive, is only $f_{V|T}(v = 1|t = 1) = 12.2\%$, which is very surprising. This low percentage can mostly be attributed to the low specificity of the test, which leads to many false positive results.

In exercise EE you'll be asked to compute the probability a patient has the virus given the result of the diagnostic test is negative, $f_{V|T}(v|t = 0)$, using Bayes' rule. In problem PP you'll be asked to analyze a different diagnostic test S with much higher specificity (98%), and observe how the probability $f_{V|S}(v = 1|s = 1)$ is affected.

Multivariable expectations

Recall the notion of expected value $\mathbb{E}_X[w(X)] \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} w(x) \cdot f_X(x)$, which evaluates the weighted average of the function $w(X)$ over all possible outcomes of the random variable X .

The multivariable expectation formula is similarly defined. Given any function $w(X, Y)$ which depends on the random variables X and Y , the expected value of $w(X, Y)$ under the probability mass function f_{XY} is defined as the summation:

$$\mathbb{E}_{XY}[w(X, Y)] \stackrel{\text{def}}{=} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} w(x, y) \cdot f_{XY}(x, y).$$

The sample space of f_{XY} is two-dimensional, so we must sum over all possible pairs of values (x, y) from the sample space $\mathcal{X} \times \mathcal{Y}$.

Multivariable expectation has the following properties, which are similar to the properties of single-variable expectations:

- $\mathbb{E}_{XY}[c] = c$
- $\mathbb{E}_{XY}[\alpha \cdot g(X, Y)] = \alpha \cdot \mathbb{E}_{XY}[g(X, Y)]$
- $\mathbb{E}_{XY}[g(X, Y) + h(X, Y)] = \mathbb{E}_{XY}[g(X, Y)] + \mathbb{E}_{XY}[h(X, Y)]$

The latter two properties can be combined, to show the expectation operator \mathbb{E}_{XY} obeys the *linear property*:

$$\mathbb{E}_{XY}[\alpha g(X, Y) + \beta h(X, Y)] = \alpha \mathbb{E}_{XY}[g(X, Y)] + \beta \mathbb{E}_{XY}[h(X, Y)].$$

Additionally, we have the following simplification formulas when computing expectations of functions that depend on only one of the two variables:

- $\mathbb{E}_{XY}[g(X)] = \mathbb{E}_X[g(X)]$
- $\mathbb{E}_{XY}[h(Y)] = \mathbb{E}_Y[h(Y)]$

The first formula holds because the randomness in $g(X)$ doesn't depend on Y , so \mathbb{E}_{XY} is equivalent to \mathbb{E}_X . Similarly, $\mathbb{E}_{XY} = \mathbb{E}_Y$ when computing the expectation of $h(Y)$, which doesn't depend on X .

The expected value of a sum of random variables $X + Y$ is the sum of their expectations:

$$\mathbb{E}_{XY}[X + Y] = \mathbb{E}_{XY}[X] + \mathbb{E}_{XY}[Y] = \mathbb{E}_X[X] + \mathbb{E}_Y[Y].$$

The first equality follows from the linear property of the expectation operator. The second equality follows from the simplification formulas $\mathbb{E}_{XY}[g(X)] = \mathbb{E}_X[g(X)]$, and $\mathbb{E}_{XY}[h(Y)] = \mathbb{E}_Y[h(Y)]$, which we just learned about.

The main reason we're learning about the expectation operator is because it is used to define the covariance operation, which we'll see in the next section.

Covariance and correlation

Recall the formula $\mathbf{var}(X) = \mathbb{E}_X[(X - \mu_X)^2]$, which computes the variance of a single random variable X . The *covariance* is a generalization of the variance formula defined for two random variables:

$$\begin{aligned} \mathbf{cov}(X, Y) &= \mathbb{E}_{XY}[(X - \mu_X)(Y - \mu_Y)] \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (x - \mu_X)(y - \mu_Y) f_{XY}(x, y), \end{aligned}$$

where $\mu_X = \mathbb{E}_X[X]$ and $\mu_Y = \mathbb{E}_Y[Y]$ are the means of the marginal distributions f_X and f_Y . In words, the covariance $\mathbf{cov}(X, Y)$ measures the joint variability of two random variables X and Y . The covariance formula obeys the following properties:

- $\mathbf{cov}(X, X) = \mathbf{var}(X)$
- $\mathbf{cov}(X, Y) = \mathbf{cov}(Y, X)$
- $\mathbf{var}(X + Y) = \mathbf{var}(X) + \mathbf{var}(Y) + 2\mathbf{cov}(X, Y)$
- $\mathbf{cov}(X, a) = 0$
- $\mathbf{cov}(X + c, Y + d) = \mathbf{cov}(X, Y)$
- $\mathbf{cov}(aX, bY) = ab \mathbf{cov}(X, Y)$
- $-\mu_X \mu_Y \leq \mathbf{cov}(X, Y) \leq \mu_X \mu_Y$

Using the properties of expectations, we can rewrite the covariance formula as follows:

$$\begin{aligned} \mathbf{cov}(X, Y) &= \mathbb{E}_{XY}[(X - \mu_X)(Y - \mu_Y)] \\ &= \mathbb{E}_{XY}[XY - X\mu_Y - Y\mu_X + \mu_X\mu_Y] \\ &= \mathbb{E}_{XY}[XY] - \mathbb{E}_{XY}[X]\mu_Y - \mathbb{E}_{XY}[Y]\mu_X + \mu_X\mu_Y \\ &= \mathbb{E}_{XY}[XY] - \mu_X\mu_Y - \mu_Y\mu_X + \mu_X\mu_Y \\ &= \mathbb{E}_{XY}[XY] - \mu_X\mu_Y. \end{aligned}$$

In words, this calculation tells us the covariance $\mathbf{cov}(X, Y)$ can be computed as the expectation of the product XY minus the product of the means of the marginals.

Thus we have an alternative formula for calculating the covariance:

$$\mathbf{cov}(X, Y) = \mathbb{E}_{XY}[XY] - \mu_X \mu_Y,$$

which is can be easier to compute, since it requires only computing the single expectation.

The *correlation* between the random variables X and Y is denoted $\mathbf{corr}(X, Y)$ or ρ_{XY} . The correlation between X and Y is defined as the ratio of the covariance $\mathbf{cov}(X, Y)$ to the product of the variables' standard deviations:

$$\mathbf{corr}(X, Y) = \frac{\mathbf{cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Dividing the covariance by the product of the standard deviations $\sigma_X \sigma_Y$ has a normalizing effect, constraining the correlation $\mathbf{corr}(X, Y)$ to always be between -1 and 1 .

Recall we've already seen the concepts of covariance and correlation earlier in Section ??, when we learned about descriptive statistics. We measured association between two variables x and y in a dataset $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, by computing the $\mathbf{Cov}(x, y) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \mathbf{Mean}_x)(y_i - \mathbf{Mean}_y)$ and $\mathbf{Corr}(x, y) = \frac{\mathbf{Cov}(x, y)}{s_x s_y}$. The difference is that we're now computing covariance and correlation based on probability distributions that represent all possible observations from the probability distribution f_{XY} , instead of one particular dataset of observations.

The correlation $\mathbf{corr}(X, Y)$ measures the strength of the association between the variables X and Y , under the assumption that the underlying dependence is a *linear* relationship. We'll learn more about modelling linear relationships between random variables in Chapter ??.

* * *

The formulas and equations we showed in this section apply to all joint probability distribution of multiple random variables. In the next section, we'll discuss an important special case of independent random variables, and see how many of the formulas and equations become simpler when random variables are independent.

2.2.5 Independent random variables

Two random variables X and Y are called *independent* if their joint probability distribution can be written as the product of two independent distributions:

$$f_{XY}(x, y) = f_X(x)f_Y(y).$$

Intuitively, the randomness of X does not depend on the randomness of Y , and vice versa. This means all probability calculations with the pair of random variables (X, Y) can be simplified by analyzing X and Y separately.

When X and Y are independent random variables, the conditional distribution function $f_{Y|X}$ is equal to the marginal distribution f_Y . In other words, knowing the value of X doesn't change anything about our uncertainty of Y . Similarly, the conditional distribution $f_{X|Y}$ is equal to the marginal f_X .

Formulas for independent random variables

Let's revisit the expectations and covariance formulas for two random variables X and Y , in the special case when X and Y are independent.

The expected value of the product XY is the product of their expectations:

$$\mathbb{E}_{XY}[XY] = \mathbb{E}_X[X] \cdot \mathbb{E}_Y[Y]. \quad (\text{when } X \text{ and } Y \text{ independent})$$

This follows because the X -randomness is independent of the Y -randomness, so we can split the two expectations.

If we substitute this result into the general formula for covariance $\text{cov}(X, Y)$, we obtain

$$\begin{aligned} \text{cov}(X, Y) &= \mathbb{E}_{XY}[XY] - \mu_X \mu_Y \\ &= \mathbb{E}_X[X] \cdot \mathbb{E}_Y[Y] - \mu_X \mu_Y \quad (\text{when } X \text{ and } Y \text{ independent}) \\ &= \mu_X \mu_Y - \mu_X \mu_Y = 0, \end{aligned}$$

which tells us the covariance of two independent random variables is zero. The third equation follows from the definitions of the means: $\mu_X \stackrel{\text{def}}{=} \mathbb{E}_X[X]$ and $\mu_Y \stackrel{\text{def}}{=} \mathbb{E}_Y[Y]$.

Since the correlation coefficient $\text{corr}(X, Y)$ is proportional to the covariance $\text{cov}(X, Y)$, the correlation coefficient is also zero:

$$\text{corr}(X, Y) = 0. \quad (\text{when } X \text{ and } Y \text{ independent})$$

Another useful formula we can obtain states that the variance of the sum of two independent variables is the sum of their variances:

$$\mathbf{var}(X + Y) = \mathbf{var}(X) + \mathbf{var}(Y). \quad (\text{when } X \text{ and } Y \text{ independent})$$

This follows from the general formula for $\mathbf{var}(X + Y)$ that we saw earlier, $\mathbf{var}(X + Y) = \mathbf{var}(X) + \mathbf{var}(Y) + 2\mathbf{cov}(X, Y)$ ⁰, where the last term is zero because $\mathbf{cov}(X, Y) = 0$ when X and Y are independent.

Multiple independent, identically distributed variables

Let's generalize what we learned about two independent random variables X and Y , to the case of n independent random variables X_1, X_2, \dots, X_n . We'll assume that each of the random variables X_i is an instance of the same probability distribution f_X .

The analysis of sequences of random variables (X_1, X_2, \dots, X_n) , where each $X_i \sim f_X$, is very important for statistics, so it's worth taking a closer look at the formulas we can derive for describing sequences of n independent copies of the random variable X .

Since all the X_i s are independent, the joint probability distribution for the sequence (X_1, X_2, \dots, X_n) can be written as the product of n copies of the probability distribution of the random variable X :

$$f_{X_1 X_2 \dots X_n}(x_1, x_2, \dots, x_n) = f_X(x_1)f_X(x_2) \cdots f_X(x_n).$$

We call this the *independent, identically distributed* setting, or *i.i.d.* for short. This *product structure* of the joint distribution $f_{X_1 X_2 \dots X_n}$ tells us the random variables are *independent*, and each X_i is an *identical* copy of the random variable $X \sim f_X$, so the name *i.i.d.* is suitable.

Let's now talk about one particular observation of the sequence of random variables (X_1, X_2, \dots, X_n) , which we'll denote (x_1, x_2, \dots, x_n) . Each x_i is a particular observation from the random variable X , and we have a sequence of n such observations. We'll sometimes refer to the sequence (x_1, x_2, \dots, x_n) as a *sample*, so we don't have to say "sequence of n independent observations from the random variable X ."

Another way to describe the sample (x_1, x_2, \dots, x_n) is a draw from the joint probability distribution $f_{X_1 X_2 \dots X_n}$. To generate such a sample, we make n observations from the random variable X .

Example 5: tossing a coin n times Recall the random variable C that describes the outcome of a coin toss. We'll identify the outcomes of the coin toss as 0 for tails and 1 for heads. The sample space the random variable C is therefore $\mathcal{C} = \{0, 1\}$. If the coin is fair, the probability mass function of the random variable C will have the

values $f_C(1) = \frac{1}{2}$ and $f_C(0) = \frac{1}{2}$. See Figure 2.7 for an illustration of the probability mass function of the random variable C .

We can describe the process of tossing the coin n times as a sequence of n copies of the random variable C : (C_1, C_2, \dots, C_n) . Each copy C_i describes the outcome of one coin toss, that can be either 0 or 1 with probability $\frac{1}{2}$.

We denote a particular outcome of the process using lowercase letters (c_1, c_2, \dots, c_n) , where each of the outcomes c_i is either 0 or 1. Note the difference between the two concepts: (C_1, C_2, \dots, C_n) is a mathematical model that describes all possible outcomes of the experiment, while (c_1, c_2, \dots, c_n) describes the particular outcomes observed for one experiment.

Suppose we now define a new variable S that describes the sum of the random variables in the sequence (C_1, C_2, \dots, C_n) :

$$S = \sum_{i=1}^n C_i = C_1 + C_2 + \dots + C_n.$$

In words, S represents the count of number of heads after n coin tosses. The sample space of the random variable S is $\{0, 1, 2, \dots, n\}$, since we can observe anywhere from 0 to n heads outcomes in n coin tosses.

If we're interested in knowing the mean of the random variable S , we need to compute $\mu_S = \mathbb{E}_S[S]$. We know the random variable S is defined as the sum of an i.i.d. sequence of the random variable C , so we can use the properties of expectations we learned above to obtain μ_S :

$$\begin{aligned} \mu_S = \mathbb{E}_S[S] &= \mathbb{E}_{C_1 C_2 \dots C_n} [C_1 + C_2 + \dots + C_n] \\ &= \mathbb{E}_C [C_1] + \mathbb{E}_C [C_2] + \dots + \mathbb{E}_C [C_n] \\ &= n \mathbb{E}_C [C] \\ &= n (1 \cdot f_C(1) + 0 \cdot f_C(0)) \\ &= n \left(1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2} \right) = \frac{n}{2} \end{aligned}$$

The second equality follows from the independence assumption, while the third equality follows from the identical assumption.

We can also compute the variance $\sigma_S^2 = \text{var}(S)$ relying on the i.i.d. assumption. See Exercise E2.17.

Example 6: rolling a die n times Consider the sequence n rolls of a fair die (D_1, D_2, \dots, D_n) , where each D_i is an independent copy of the die roll random variable D that we saw earlier in Example 2. See Figure 2.8 for an illustration of the probability mass function of the random variable D .

Define the random variable A which computes the average value of the sequence (D_1, D_2, \dots, D_n) :

$$A = \frac{1}{n} \sum_{i=1}^n D_i = \frac{1}{n} [D_1 + D_2 + \dots + D_n].$$

We can use the i.i.d. properties of A to obtain its mean μ_A and its variance σ_A^2 . See exercise E2.18 and E2.19.

* * *

The analysis of the properties of sequences of n independent observations from a random variable (X_1, X_2, \dots, X_n) as shown in the above examples serve as “foreshadowing” for the ideas we’ll discuss in the rest of the chapter. In Section 2.8 we’ll study the properties of i.i.d. scenario in more details, and state the *central limit theorem*, which is an important, foundational result that describes the properties of random samples, and serves as a foundation for statistics.

2.2.6 Discussion

In this section, we learned how to model multiple random variables X, Y using a joint probability distribution f_{XY} . We can subdivide the multivariable probability modelling task into two categories:

- Modelling *independent* random variables, like the coin tosses in Example 1 and Example 5, the die rolls in Example 2 and Example 6.
- Modelling *dependent* random variables, like the outcomes of the coin-die game in Example 3, and the medical diagnostic test in Example 4.

It’s important for you to keep in mind that these two categories describe fundamentally different scenarios, and different formula apply in each case.

Let’s summarize the facts that we know about the probability distributions and formulas for the two categories:

Independent random variables Scenarios with independent random variables X and Y are in some sense the “easy case,” because the joint probability distribution has the form of the product of the marginals. There is no need to use conditional distributions $f_{Y|X}$, since the randomness in Y doesn’t depend on the randomness in X .

Using the linear property of expectations and the linearity of variance allows us to do calculations with sums of random variables, like the random variables S from Example 5 and A from Example 6.

The covariance $\text{cov}(X, Y)$ of two independent random variables is zero, and by extension their correlations is also zero $\text{corr}(X, Y) = 0$.

Dependent random variables When two random variables X and Y are not independent, we call them *dependent* or *correlated* random variables. This describes the “general case” when the joint probability distribution f_{XY} cannot be written as the product of two independent distributions for each random variable.

It’s important to keep in mind that none of the formulas we derived for independent random can be used in the general case when X and Y are not independent. To avoid any possible confusion, equation in this section that apply only for independent random variables are clearly marked with the phrase “when X and Y are independent.”

The chain rule allows us to write the joint probability distribution f_{XY} in terms of the marginal f_X and the conditional distribution $f_{Y|X}$, or in terms of the marginal f_Y and the conditional distribution $f_{X|Y}$.

The *covariance* and *correlation* formulas can be used as a crude measure of the strength of the dependence between two random variables.

Exercises

E2.17 Compute the variance of $S = \sum C_i$

E2.18 Compute the mean of $A = \sum D_i$

E2.19 Compute the variance of $A = \sum D_i$

TODO: add example of uncorrelated but not independent RVs <https://stats.stackexchange.com/questions/85363/simple-examples-of-uncorrelated-but-not-independent-x-and-y>

Links

[Relevant pages from Wikipedia]

https://en.wikipedia.org/wiki/Joint_probability_distribution

https://en.wikipedia.org/wiki/Marginal_distribution

https://en.wikipedia.org/wiki/Conditional_probability_distribution

[Discussion on the Bayesian way of thinking about probabilities]

https://en.wikipedia.org/wiki/Bayesian_probability

[Condensed review of all of probability theory]

<https://ermongroup.github.io/cs228-notes/preliminaries/probabilityreview/>

[The Gambler's fallacy explained by Joe Bertolami]

<http://bertolami.com/index.php?engine=blog&content=posts&detail=probability>

[Bayes' Theorem explained with LEGO]

<http://www.countbayesie.com/blog/2015/2/18/bayes-theorem-with-lego>

[The geometry Bayes theorem explained by 3Blue1Brown]

<https://www.youtube.com/watch?v=HZGCoVF3YvM>

[An article about the sensitivity and specificity of diagnostic tests]

<https://www.ncbi.nlm.nih.gov/books/NBK557491/>

[Chain rule of probability theory]

[https://en.wikipedia.org/wiki/Chain_rule_\(probability\)](https://en.wikipedia.org/wiki/Chain_rule_(probability))

2.3 Inventory of discrete distributions

It's time to introduce the probability distributions used in statistics. We've already seen some examples involving the uniform distribution $\mathcal{U}_d(\alpha, \beta)$ and the Poisson distribution $\text{Pois}(\lambda)$, but there are a couple of others that you should know about. Recall the notation $X \sim \mathcal{M}(\theta)$, which we use to denote the random variable X distributed according to the model family \mathcal{M} initialized with parameters θ . In this section, we'll learn about the different probability models \mathcal{M} that exist and the different choices for their parameters θ .

Figure 2.27 shows examples of the different shapes of the discrete probability distributions families \mathcal{M} that we'll discuss in this section.

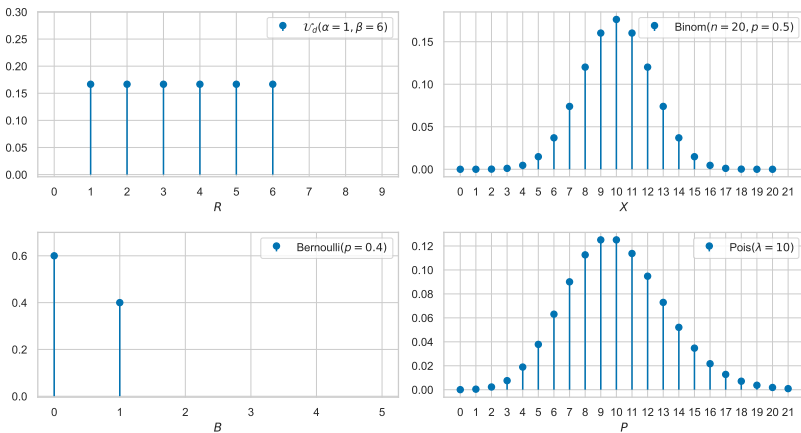


Figure 2.27: Graphs of the probability mass functions of four common discrete probability distributions: uniform, binomial, Bernoulli, and Poisson.

The four discrete probability distributions that are most commonly used in statistics are:

- The *discrete uniform* distribution $\mathcal{U}_d(\alpha, \beta)$, which assigns equal probabilities to the integers between α and β , inclusively.
- The *Bernoulli* distribution $\text{Bernoulli}(p)$, which describes the coin toss of a biased coin, for which the outcome “success”/heads/1 has probability p , and the outcome “failure”/tails/0 has probability $(1 - p)$.
- The *binomial* distribution $\text{Binom}(n, p)$, which describes the number of successes in a sequence of n Bernoulli trials.
- The *Poisson* distribution $\text{Pois}(\lambda)$, which models the number of times an event occurs in some interval, given that the average number of occurrences is λ .

All these distributions will come up in one statistical analysis scenario or another later in the book, so it's a good idea to get to know them now. You're not expected to memorize any of the definitions or equations associated with these distributions, but rather think of this section as a "reference manual" you can return to later when you need to use one of these functions.

We'll also briefly talk about some other distributions (marked as "(optional)" in the table of contents), which are less important since they are not often used in statistics. Examples of this "optional reading material" includes the discussion on the *geometric* distribution $\text{Geom}(p)$ (waiting time until the first success in a series of Bernoulli trials), the *negative binomial* distribution $\text{NBinom}(r, p)$ (waiting time until r successes in series of Bernoulli trials), and the *hypergeometric* distribution $\text{Hypergeom}(a, b, n)$. These are all interesting to see once, but they won't play a big rest of the book. We've only included them to make this section a *complete* inventory of all discrete probability distributions, in case you want to play with these probabilistic LEGOs later on.

Before we get to the list of distributions, let's take a moment to introduce some math concepts that we'll need for the rest of the section.

2.3.1 Math prerequisites

In this section, we'll learn some counting and summations tricks that will help us understand discrete probability distributions. We'll start with *combinatorics* formulas, which are used to for counting the number of ways certain events can occur. Combinatorics concepts are useful to know because they are used in the definitions of different discrete probabilities distributions. We'll then show some useful summation formulas for computing the probability of composite outcomes.

Introduction to combinatorics

We'll now discuss some important formulas for counting the number of ways that certain events can occur. Thinking in terms of "how many ways can you choose x from the set Y " allows us to compute all kinds of probabilities, so mathematicians developed some tools and notation for this type of calculations: factorials, combinations, permutations, etc.

For example, how many ways can you choose a two-card hand of poker from a set of 52 cards. In this example, x is a two-card poker "hand", Y is a standard deck of 52 cards $\{1, 2, 3, \dots, 52\}$ where

1 corresponds to A spades, and 52 is two of clubs. Currently, this may seem like a complicated calculation to do, but by the end of this section you'll know the answer is $\binom{52}{2}$, where the weird-looking notation is pronounced "52 choose 2." You'll also learn there is a math formula $\frac{n!}{k!(n-k)!}$ for computing $\binom{n}{k}$, the "number of ways you can choose k items from a set of n items, when the order doesn't matter."

Why are we learning this stuff? In order to motivate you to read the next 10 pages of dense math formulas, I should probably explain why the idea of counting the number of ways a given outcome can occur is such a useful concept. Think about it: if mathematicians spent time developing specialized notation for this stuff, it's probably useful.

Many probability scenarios can be described a sample space of equally likely outcomes. Computing probabilities of some composite outcome A is just a matter of counting the number of ways a certain outcome can occur. then dividing by the total number of possible outcomes:

$$\Pr(\{\text{outcome } A\}) = \frac{\text{number of ways outcome } A \text{ can occur}}{\text{total number of possible outcomes}}.$$

Basically, if you learn these 10 pages of "boring" and "scary" looking math formulas for counting "number of ways ..." in various scenarios, you'll have a solid foundation for understanding discrete probability distributions.

Definitions and notation

There are four formulas for counting configurations that you need to be familiar with:

- $n_1 \cdot n_2$: the product of the number of independent outcomes.
- $n!$: the *factorial* function that counts the number of ways we can order a list of n objects.
- ${}_nP_k$: the number of *permutations* of k objects chosen from a set of n objects.
- ${}_nC_k = \binom{n}{k}$: the number of *combinations* of k objects chosen from a set of n objects.

We'll now show the formulas and the logic behind these four formulas for combinatorial calculations. Instead of trying to memorize the formulas, it's easier to remember the "procedure" behind each of these operations.

Products If a composite outcome can be described by a sequence of k steps, where the number of ways of outcomes in step i is n_i , then the total number of ways of different outcomes that can occur is the product $n_1 \cdot n_2 \cdots n_k$.

Recall the product rule of probabilities, which we use used to describe independent outcomes: $f_{X_1 X_2} = f_{X_1} \cdot f_{X_2}$. If $f_{X_1}(x_1) = \frac{1}{n_1}$ (n_1 possible outcomes with equal probability), and $f_{X_2}(x_2) = \frac{1}{n_2}$ (n_2 equiprobable outcomes), then $f_{X_1 X_2}(x_1, x_2) = \frac{1}{n_1} \cdot \frac{1}{n_2}$.

Factorials The factorial of the integer n is defined as the product of all positive integers less than or equal to n :

$$n! \stackrel{\text{def}}{=} n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1.$$

The factorial function $n!$ describes the number of ways of arranging n distinct objects into an ordered sequence. When choosing which object to put first in the list, we can select from n candidates. Then to select the second object, we can choose from the remaining $(n-1)$ candidates, and when choosing the third object we have $(n-2)$ possibilities. This process continues until the last object of the list, when we have only one choice. For example, the number of ways you can order four elements is $4! = 4 \cdot 3! = 4 \cdot 6 = 24$.

You can compute $n!$ using pen and paper when n is a small number. For larger values of n we can use the computer.

```
>>> from scipy.special import factorial
>>> factorial(4) # 4! = 4*3*2*1
24
```

code
2.3.1

Note the factorial function grows very quickly:

```
>>> [factorial(k) for k in [5,6,7,8,9,10,11]]
[120, 720, 5040, 40320, 362880, 3628800, 39916800]
```

code
2.3.2

The factorial $15!$ is more than 1.3 trillion:

```
>>> factorial(15)
1307674368000
```

code
2.3.3

The factorial function is the basis for the definition of the permutations and the combinations formulas, which we'll discuss next.

Permutations The math term *permutation* refers to the different possible arrangement of a list of elements into a sequence. The number of permutations of k objects selected from a list of n objects is given by the formula:

$${}_n P_k = \underbrace{n \cdot (n-1) \cdot (n-2) \cdots (n-k+1)}_{k \text{ factors}} = \frac{n!}{(n-k)!}.$$

In words, the quantity ${}_nP_k$ describes the product of all positive integers between n and $(n - k + 1)$. The logic behind this formula is very similar to the counting we for the factorial, but the product contains only k factors, since we are selecting only k objects.

The second equality follows from the following calculation

$$\begin{aligned}\frac{n!}{(n-k)!} &= \frac{n \cdot (n-1) \cdots (n-k+1) \cdot (n-k) \cdot (n-k-1) \cdots 3 \cdot 2 \cdot 1}{(n-k) \cdot (n-k-1) \cdots 3 \cdot 2 \cdot 1} \\ &= n \cdot (n-1) \cdots (n-k+1) \frac{(n-k) \cdot (n-k-1) \cdots 3 \cdot 2 \cdot 1}{\cancel{(n-k) \cdot (n-k-1) \cdots 3 \cdot 2 \cdot 1}} \xrightarrow{1} \\ &= n \cdot (n-1) \cdots (n-k+1).\end{aligned}$$

In other words, the effect of dividing the factorial $n!$ by the factorial $(n - k)!$ has the effect of cancelling all factors $(n - k)$ and smaller from the calculation.

Let's look at a concrete example to understand how the formula works. The number of permutations of 2 items selected from a list of 5 items is

$${}_5P_2 = \frac{5!}{(5-2)!} = \frac{5!}{3!} = \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{3 \cdot 2 \cdot 1} = 5 \cdot 4 = 20.$$

Dividing by $3!$ has the effect of stopping the factorial calculation $5!$ after two steps. We can use the function `perm(n,k)` defined in the module `scipy.special` to compute the same answer.

```
>>> from scipy.special import perm
>>> perm(5,2)
20
```

code
2.3.4

The function `perm` only tells us the number of permutations. If we want to actually see the list of these 20 permutations, we can use the function `permutations` from the module `itertools`:

```
>>> from itertools import permutations
>>> n = 5
>>> nitems = range(1,n+1)
>>> k = 2
>>> list(permutations(nitems, k))
[(1,2), (1,3), (1,4), (1,5), (2,1), (2,3), (2,4), (2,5),
 (3,1), (3,2), (3,4), (3,5), (4,1), (4,2), (4,3), (4,5),
 (5,1), (5,2), (5,3), (5,4)]
```

code
2.3.5

Note the list contains both the permutations $(1, 2)$ and $(2, 1)$, since the order of the k elements chosen matters. How many ways are there of selecting k elements from an n -element set, when the order doesn't matter? You're about to find out!

Combinations Let's now think about the number of *combinations* we can make by selecting a *set* of k objects from a list of n objects.

The order of the k selected items doesn't matter. The formula that describes the number of combinations of k objects selected from a list of n items is:

$${}_nC_k \stackrel{\text{def}}{=} \frac{n!}{(n-k)!k!}.$$

We often use the alternative notation $\binom{n}{k}$, read " n choose k ," to denote the number of combinations of size k selected from n possible items.

We can understand this formula as a modification on the formula for permutations we saw in the previous section.

$${}_nC_k = \frac{{}_nP_k}{(n-k)!}.$$

Permutations are sequences of the form (a_1, a_2, \dots, a_k) , where the order of the elements matters. To obtain the number of sets of the form $\{a_1, a_2, \dots, a_k\}$, we need to divide the number of permutations by the number of possible ways to sort k elements, which is given by the factorial $k!$.

For example, the number of combinations of size 2 selected from a list of 5 items is

$${}_5C_2 = \binom{5}{2} = \frac{5!}{(5-2)!2!} = \frac{5!}{(3!2!)} = \frac{5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{3 \cdot 2 \cdot 1 \cdot 2 \cdot 1} = \frac{20}{2} = 10.$$

Dividing by $3!$ has the effect of stopping the factorial calculation $5!$ after two steps, and dividing by $2!$ is because we don't care about the order of the two elements selected. The function `comb(n,k)` defined in the module `scipy.special` gives the same answer:

```
>>> from scipy.special import comb
>>> comb(5,2) # = "n choose k"
10
```

code
2.3.6

To see a list of all possible combinations of $k = 2$ items selected from a set of $n = 5$ items, use the function `permutations` from the module `itertools`:

```
>>> from itertools import combinations
>>> n = 5
>>> nitems = range(1,n+1)
>>> k = 2
>>> list(combinations(nitems, k))
[(1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4),
 (3,5), (4,5)]
```

code
2.3.7

Note this list contains the combination $(1,2)$ which corresponds to the set $\{1,2\}$, and doesn't contain $(2,1)$ since it corresponds to the same set $\{1,2\}$. Compare the list of combinations with the list of permutations we obtained earlier in code block 2.3.5.

Example Let's look at a real-world situation that in which we use the four different ways of counting configurations. Tracy is launching a new website. She has come up with a number of different alternatives for the main heading text and the subheading text. For the company blog (which is also in scope somehow!), she has prepared five blog posts, and also has seven user testimonials that say how great her company's products are.

Instead of making a single choice for the main heading text, the subheading text, the blog posts and the user testimonials to show to website, Tracy wants to make a "random" website that shows a different page elements to each visitor. Why choose when you can randomize?

Before building her "random webpage" idea, she wants to do some basic checks to see how many different choices for the websites elements there will be for various configurations.

1. Suppose she has 2 candidates for the main heading text and 3 choices for the subheading text. The number of different choices for main heading text plus subheading text is given by the product

$$2 \times 3 = 6.$$

We have two alternatives choices for the main heading and three alternatives for the subheading, so there are six possible choices for the page header.

2. Tracy has prepared 5 blog posts and wants to show all of them in the "from our blog" section of the website. The order of appearance matters. The number of different ways to sequence the "blog posts" section of the website are given by the factorial:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120.$$

There are five choices for the first blog post, then four choices for which blog post will appear second, three remaining choices for the third blog post, two choices for the fourth, and only one choice for the last.

3. Suppose now that Tracy decides she only wants to show two posts instead of all five—she doesn't want to overwhelm visitors with too much text. The number of different ways to build the "from our blog" section of the website is then

$${}_5P_2 \stackrel{\text{def}}{=} \frac{5!}{3!} = \frac{5 \times 4 \times 3 \times 2 \times 1}{3 \times 2 \times 1} = 5 \times 4 = 20.$$

The reasoning is similar to the previous case (five choices for the first article and four choices for the second article), but we stop the factorial calculation after two steps since she only wants to show two blog posts.

4. *Combinations* are used to count the number of outcomes where the order of the events doesn't matter. Tracy has collected a total of 7 user testimonials, but only wants to show three user testimonials on the homepage. The user testimonials will appear side-by-side as three columns, and we assume the order of appearance doesn't matter. The number of ways to build the "user testimonials" section is

$${}_7C_3 \stackrel{\text{def}}{=} \binom{7}{3} \stackrel{\text{def}}{=} \frac{7!}{3!4!} = 35.$$

The logic for the combinations calculations is similar to the formula for permutations, but an extra factor of $3!$ appears in the denominator since the order doesn't matter.

Combinatorial formulas

We arbitrarily define the factorial of zero to be one $0! = 1$, because this will help write certain formulas more easily. For example, the number of ways to choose zero elements is defined as one:

$$\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{1 \cdot n!} = 1.$$

The number of ways to choose n elements from a set of n is also one;

$$\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n! \cdot 0!} = 1.$$

Here are some other formulas and relations for the number of combinations:

$$\binom{n}{k+1} = \binom{n}{k} \frac{n-k}{k+1}.$$

The recursive formula is

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

$$\binom{M}{n} = \frac{M}{n} \binom{M-1}{n-1}$$

Don't worry too much about these formulas. We're including them here in case you might need one of them when solving the exercises.

Summations formulas

Recall that computing the probability of the event $\{a \leq X \leq b\}$ is defined as the summation of the values of the probability mass function $f_X(x)$ for x varying from $x = a$ until $x = b$:

$$\Pr(\{a \leq X \leq b\}) = \sum_{x=a}^{x=b} f_X(x) = f_X(a) + f_X(a+1) + \cdots f_X(b).$$

Here are some formulas for calculating the sum of sequences $a_k = k$ and $b_k = k^2$. The formulas for the sum of the first N positive integers is

$$\sum_{k=1}^N k = \frac{N(N+1)}{2}.$$

The sum of the *squares* of the first N positive integers is

$$\sum_{k=1}^N k^2 = \frac{N(N+1)(2N+1)}{6}.$$

Sum of cubes of the first N positive integers is

$$\sum_{k=1}^N k^3 = \left(\frac{N(N+1)}{2} \right)^2.$$

Let's verify these formulas, but computing the sum of the first $N = 10$ integers using Python. First we'll create the list of numbers named `nums`, which contains the numbers from 1 until $N = 10$.

```
>>> N = 10
>>> nums = range(1,N+1)
>>> nums
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

code
2.3.8

We can compute the sum of the values in the list `nums` using the Python built-in function `sum`:

```
>>> sum(nums)
55
```

code
2.3.9

We can verify the formula $\frac{N(N+1)}{2}$ gives the same number when $N = 10$.

The sum of the squares of the numbers in the `list` is obtained as follows:

```
>>> squarednums = [num**2 for num in nums]
>>> squarednums
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> sum(squarednums)
385
```

code
2.3.10

We can obtain the same answer using the formula $\frac{N(N+1)(2N+1)}{6}$ when $N = 10$.

The sum of the cubes of the numbers is:

```
>>> cubednums = [num**3 for num in nums]
>>> cubednums
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
>>> sum(cubednums)
3025
```

code
2.3.11

This is what we obtain from the formula $\left(\frac{N(N+1)}{2}\right)^2$ when $N = 10$.

The geometric series

Since we're on the topic of summations, it would be a good moment to introduce the *geometric series*, which describes a summation of terms in the geometric sequence $(1, r, r^2, r^3, r^4, \dots)$. The pattern in the geometric sequence is that each term is r -times the previous term. The formula for the k^{th} term in the geometric sequence is $c_k = r^k$.

If $|r| < 1$, there is an exact formula for the sum of the geometric series:

$$\sum_{k=0}^{\infty} r^k = 1 + r + r^2 + r^3 + r^4 + \dots = \frac{1}{1-r}.$$

This is pretty crazy if you think about it! The expression on the left contains infinitely many terms, yet somehow, the sum of all of these terms is described by a simple mathematical formula $\frac{1}{1-r}$.

More generally, the terms in the geometric series could contain a multiplicative constant a , in which case the summation formula becomes:

$$\sum_{k=0}^{\infty} ar^k = a + ar + ar^2 + ar^3 + ar^4 + \dots = \frac{a}{1-r}.$$

Example Let's compute the sum of the geometric series with $a = \frac{1}{2}$ and $r = \frac{1}{2}$. Applying the above formula, we obtain

$$\sum_{k=0}^{\infty} \frac{1}{2} \left(\frac{1}{2}\right)^k = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots = \frac{\frac{1}{2}}{1 - \frac{1}{2}} = 1.$$

Figure 2.28 shows a visualization of how the numbers in this series add up to give the total of 1.

We can also compute the sum of the geometric series using Python code. Instead of computing infinitely many terms, we'll stop the summation after 60 terms because higher terms are negligibly small.

code
2.3.12

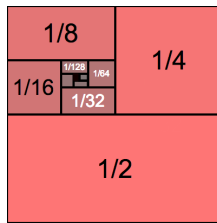


Figure 2.28: A graphical representation of the infinite sum of the geometric series $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \dots$. The area of each region corresponds to one of the terms in the series. The total area is equal to $\sum_{k=0}^{\infty} \frac{1}{2} \left(\frac{1}{2}\right)^k = 1$.

```
>>> a = 0.5
>>> r = 0.5
>>> sum([a*r**k for k in range(0,60)])
1.0
```

* * *

At this point, you might be wondering why you had to go through all these MM pages of dense math formulas. The reason why I want to expose you to all this math, is because I wanted to expose you to the underlying “structure” in the formulas and the “stories” behind the product, factorial, permutations, and combinations calculations. This way you will never have to memorize any formulas for doing probability calculations, but instead be able to reconstruct the formula by replaying the appropriate “story” as needed. This is what learning math is all about: seeing the underlying structure and patterns in different calculations.

Exercises

E2.20 Compute the sum of $a_n = n + n^2$ for n from 0 to N .

E2.21 INSERT QUESTION requiring $\binom{n}{k}$ here...

E2.22 Compute the infinite sum ... $r=1/3$

Links

[Nice visual explanation of combinatorics formulas]

<https://www.youtube.com/watch?v=ONAASclUm4k>

[Binomial coefficient]

https://en.wikipedia.org/wiki/Binomial_coefficient

[Binomial theorem]

https://en.wikipedia.org/wiki/Binomial_theorem

[Online calculator for the permutations and combinations]

<https://www.calculatorsoup.com/calculators/discretemathematics/>

2.3.2 Review of definitions and formulas

We'll now briefly review, in condensed form, all the definitions of discrete random variables that we saw in Section 2.1. I know it has been a while, so I figured you might need a little refresher.

The *random variable* X describes a quantity that can take on different values due to uncertainty or variability. We denote random variables by uppercase letters like X , Y , and Z , and particular *outcomes* of these random variables using lowercase letters like x , y , and z . The *sample space* \mathcal{X} (calligraphic X) is the set of all possible outcomes of the random variable X . Another term to describe the sample space is *support*. In this section, we'll focus on *discrete* sample spaces, which consist of discrete values like subsets of the integers.

The *probability mass function* (pmf) is a function of the form $f_X : \mathcal{X} \rightarrow [0, 1]$, that assigns probabilities to each of the possible outcomes in the sample space of the random variable X . The probability of the outcome $\{X = x\}$ is given by the value of the probability mass function: $\Pr(\{X = x\}) = f_X(x)$.

A *composite* outcome is a set of simple outcomes, like the set of numbers between a and b : $\{a \leq X \leq b\}$, or any other subset of the sample space. The probability of a composite outcome $\{a \leq X \leq b\}$ is calculated by as the sum of the probability mass function values between a and b :

$$\Pr(\{a \leq X \leq b\}) = \sum_{x=a}^{x=b} f_X(x).$$

The *cumulative distribution function* (CDF) $F_X(b)$ describes the sum of the probabilities of all outcomes up to and including b :

$$F_X(b) \stackrel{\text{def}}{=} \Pr(\{X \leq b\}).$$

We can use the CDF F_X to compute probabilities of composite outcomes using the formula $\Pr(\{a \leq X \leq b\}) = F_X(b) - F_X(a - 1)$.

The *inverse of the cumulative distribution function* (inverse-CDF), denoted $F_X^{-1}(q)$, is used to do “inverse probability calculations.” We specify the probability $q \in [0, 1]$, and we want to find the value x_q such that the outcome $\{X \leq x_q\}$ will contain at least a proportion q of the total probability. In other words, x_q is the smallest value such that $F_X(x_q) \geq q$.

The *expectation* $\mathbb{E}_X[w(X)]$ describes the computation of the expected value of the function $w : \mathcal{X} \rightarrow \mathbb{R}$, which depends on the random variable X . Expectations are computed as the weighted sum over all possible outcomes: $\mathbb{E}_X[w(X)] \stackrel{\text{def}}{=} \sum_{x \in \mathcal{X}} w(x) f_X(x)$.

The *mean* of the random variable X is defined as the expectation $\mu_X \stackrel{\text{def}}{=} \mathbb{E}_X[X] = \sum_{x \in \mathcal{X}} x \cdot f_X(x)$. The *variance* of X is defined as $\sigma_X^2 \stackrel{\text{def}}{=} \mathbb{E}_X[(X - \mu_X)^2] = \sum_{x \in \mathcal{X}} (X - \mu_X)^2 \cdot f_X(x)$. The *standard deviation* of the random variable X is the square root of its variance.

You should already be familiar with these concepts and formulas, so if looking at any of the above formulas makes you feel uncomfortable (fills you with math anxiety), then I highly recommend you go back to Section 2.1 to review all the formulas, figures, and examples. If you skipped the exercises on your first reading, now might be a good time to try solving some of them. The best antidote to math anxiety is doing math exercises—we can't really learn math concepts by reading about them, we have to play with them.

2.3.3 Review of computer models

Let's briefly review what we learned about random variable objects `rvX` based on the computer models defined in `scipy.stats`. To create the random variable object `rvX` we must initialize a model by passing in a set of parameters: `rvX = <model>(<params>)`, where `<model>` is the name of one of the model families defined in `scipy.stats`, and `<params>` is a comma-separated list of model-specific parameters.

Once you have created the random variable object `rvX`, you can use its methods to do probability calculations. You should be already somewhat familiar with the methods available on random variable objects like `rvX` from what we learned in Section 2.1.4. But this was a long time ago, so I've compiled for you Table 2.1, which lists all the methods available on any random variable object `rvX` created from one of the families of pre-defined discrete probability distributions in `scipy.stats`.

Please keep this list in mind as you read through the rest of this section, to remind you that all the calculations you might be asked to do using math formulas can be done in one or two lines of Python code based on one of the above methods. In other words, if you know enough Python to import and initialize a model based on the pre-defined model families in `scipy.stats`, then you don't have to worry about any of the math equations!

method	args	math formula	description
<code>rvX.pmf</code>	<code>x</code>	$f_X(x)$	probability mass function
<code>rvX.cdf</code>	<code>b</code>	$F_X(b)$	cumulative dist. function
<code>rvX.ppf</code>	<code>q</code>	$F_X^{-1}(q)$	inverse of the CDF function
<code>rvX.mean</code>		$\mu_X = \mathbb{E}_X[X]$	mean of the distribution
<code>rvX.var</code>		σ_X^2	variance of the distribution
<code>rvX.std</code>		σ_X	standard deviation
<code>rvX.median</code>		$F_X^{-1}(\frac{1}{2})$	median of the distribution
<code>rvX.support</code>		\mathcal{X}	bounds of the sample space
<code>rvX.interval</code>	<code>1-a</code>	$[F_X^{-1}(\frac{\alpha}{2}), F_X^{-1}(1-\frac{\alpha}{2})]$	$(1 - \alpha)$ confidence interval
<code>rvX.rvs</code>	<code>n</code>		generate n observations from X
<code>rvX.expect</code>	<code>w</code>	$\mathbb{E}_X[w(X)]$	expected value of $w(X)$

Table 2.1: Summary of the methods of discrete random variable objects `rvX` created from one of the model families defined in `scipy.stats`.

2.3.4 Discrete distributions reference

We'll now switch the narrative to point-form listicle mode. The next NN pages contain an inventory of the most important discrete probability distributions you need to know about. The idea is to give you a quick overview of these building blocks, and provide you with a "reference manual" that you can refer to when you need to look up facts and formulas in the later chapters.

Each subsection is dedicated to one distribution family and follows the same pattern:

- Definition: What is the "story" for the generative process of the random variable?
- Formulas: What is the probability mass function f_X ? What is the formula for the mean μ_X and variance σ^2 ?
- Applications: what are the use cases for this family of probability models?
- Relations: how is the probability model related to other models. See Figure 2.38 for a graphical summary of the relations.
- Code examples: we'll show the Python code for creating a random variable object `rvX` based on one of the computer models defined in `scipy.stats`.

We'll also provide links to relevant resources where you can learn more about each distribution family.

All the complicated-looking math equations you'll encounter in the upcoming section are for "reference purposes" only—you're not expected to memorize them, just look at them as a mathematical implementation of the "story" that goes with each distribution. It's useful to see the structure of the probability density function f_X , but you'll rarely have to do calculations with it, since using one of the pre-defined computer models `scipy.stats` are already available.

Are you ready for this? Let's get started!

Discrete uniform

The *discrete uniform distribution* $\mathcal{U}_d(\alpha, \beta) = \text{randint}(\text{alpha}, \text{beta}+1)$ describes the random action of picking an integer between α and β at random. This distribution is called “uniform” since it assigns the same probability to each of the possible outcomes.

A random variable $X \sim \mathcal{U}_d(\alpha, \beta)$ has the probability mass function

$$f_X(x) = \frac{1}{\beta + 1 - \alpha} \text{ for all } x \text{ between } \alpha \text{ and } \beta.$$

The sample space is $\mathcal{X} = \{\alpha, \alpha + 1, \dots, \beta - 1, \beta\}$ and contains a total of $n = \beta + 1 - \alpha$ possible outcomes.

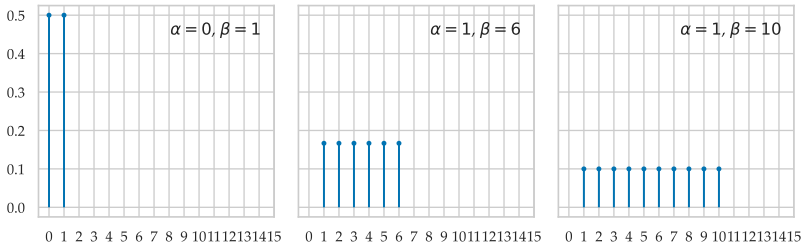


Figure 2.29: Plot of various discrete uniform distributions obtained from various choices of parameters α and β . The uniform distribution $\mathcal{U}_d(0, 1)$ is equivalent to the “coin toss” random variable C . The uniform distribution $\mathcal{U}_d(1, 6)$ is equivalent to the “die roll” random variable D . The distribution $\mathcal{U}_d(1, 10)$ describes the outcomes of rolling a 10-sided die.

The mean and variance of a random variable $X \sim \mathcal{U}_d(\alpha, \beta)$ are

$$\mu_X = \frac{\alpha + \beta}{2} \quad \text{and} \quad \sigma_X^2 = \frac{(\beta + 1 - \alpha)^2 - 1}{12}.$$

Intuitively speaking, The mean $\mu_X = \frac{\alpha + \beta}{2}$ tells us the average value of X and is the centre of α and β . For example, the mean of the random variable $D \sim \mathcal{U}_d(1, 6)$ (rolling a six sided die) is $\mu_X = \frac{1+6}{2} = 3.5$. You’ll be asked to verify the formula for the variance σ_X^2 in probrefYY.

Applications We’ve already seen many instances of the discrete uniform distribution, like the coin toss random variable C , and the roll of a six-sided die random variable D . The action of “pick a number at random between α and β ” is one of the simplest and most common type of random process, so it comes up all over the place.

Consider a sweepstakes contest with n people, and we want to award the prize to a random person. We can assign participants numeric identifiers from 1 to n , then generate an observation from the random variable $\mathcal{U}_d(1, n)$ to determine who gets the prize.

Recall the concept of *random assignment* we discussed in the DATA chapter. In a statistical experiment, if we want to split participants into an intervention group and a control group, we can define a random variable $\mathcal{U}_d(0, 1)$ and use the random observations from it to determine assignment: 0 control, and 1 intervention.

Computer models To create a computer model for the random variable $U \sim \mathcal{U}_d(\alpha, \beta)$, use the code `rvU = randint(alpha, beta+1)`. Note the extra $+1$ that we added to the second argument. Here is a complete code example:

```
>>> from scipy.stats import randint
>>> alpha = 1 # start at
>>> beta = 4 # stop at
>>> rvU = randint(alpha, beta+1)
```

code
2.3.13

Now that you have the random variable, you can call its methods to do see its properties:

```
>>> rvU.mean()
2.5
>>> rvU.var()
1.25
>>> rvU.std() # = np.sqrt(rvU.var())
1.118033988749895
```

code
2.3.14

Recall exercises E... and E..., where we obtained the formulas for the mean and variance of the discrete uniform distribution: $\mu_X = \frac{\alpha+\beta}{2}$ and $\sigma_X^2 = \frac{(\beta+1-\alpha)^2-1}{12}$. Use the formulas and the appropriate choice of α and β to verify the numerical answers obtained above.

We can obtain the limits of the sample space \mathcal{X} for the random variable `rvU` by calling its `.support()` method.

```
>>> rvU.support()
(1, 4)
```

code
2.3.15

To obtain the value of the probability mass function of f_U of the random variable `rvU`, we can use its `.pmf` method:

```
>>> for x in range(1, 4+1):
        print("f_U(", x, ") = ", rvU.pmf(x))
f_U( 1 ) = 0.25
f_U( 2 ) = 0.25
f_U( 3 ) = 0.25
f_U( 4 ) = 0.25
```

code
2.3.16

To visualize the probability mass function of the random variable `rvU` by creating a stem plot. To create a stem-plot of the probability mass function f_U , we can use the following three-step procedure:

1. Create a range of inputs xs for the plot.
2. Compute the value of $f_U = rvU$ for each of the inputs and store the results as list of values fUs .
3. Plot the values fUs at locations xs by calling the function `plt.stem(xs,fUs)`.

The code below plots the probability mass function of the random variable rvU for the range of inputs between $x = 0$ and $x = 8$:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> xs = np.arange(0, 8+1)
>>> fUs = rvU.pmf(xs)
>>> plt.stem(xs, fUs, basefmt=" ")
See stem plot shown in Figure 2.30.
```

code
2.3.17

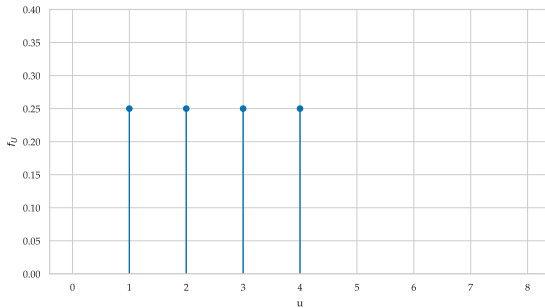


Figure 2.30: Plot of probability mass function of the random variable rvU .

Alternatively, if you don't want to manually type the above steps every time you want to look at the f_X for some rvX , you can use the helper function `plot_pmf` defined in module `plot_helpers`, which I used to generate all the plots in this book. You simply call `plot_pmf` by passing the random variable object, and the limits of the range of inputs you want to use for the plot, and it will plot the probability mass function for you:

```
>>> plot_pmf(rvU, xlims=[0,10])
Output is shown in Figure 2.30.
```

code
2.3.18

We can call the method `rvU.cdf(b)` to obtain the value of the cumulative distribution function of F_U for the random variable rvU .

```
>>> for b in range(1,4+1):
    print("F_U(", b, ") = ", rvU.cdf(b))
F_U( 1 ) = 0.25
F_U( 2 ) = 0.5
F_U( 3 ) = 0.75
F_U( 4 ) = 1.0
```

code
2.3.19

To plot the CDF, we use the three-step plot-a-function procedure with a linear space of inputs (`np.linspace`).

```
>>> import numpy as np
>>> import seaborn as sns
>>> xs = np.linspace(0,10,1000)
>>> FUs = rvU.cdf(xs)
>>> sns.lineplot(x=xs, y=FUs)
See the line plot shown in Figure 2.31.
```

code
2.3.20

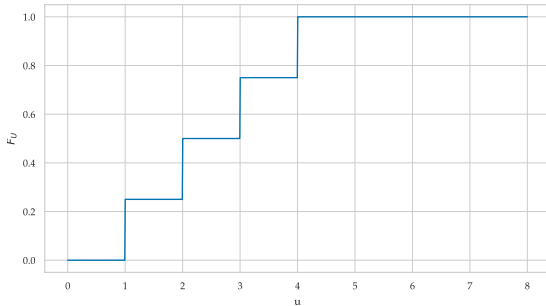


Figure 2.31: Plot of the cumulative distribution function `rvU.cdf(b)` of the continuous range of inputs b between 0 and 10. The graph of F_U starts at zero and stays zero until it jumps to $F_X(1) = \frac{1}{4}$ at $x = 1$, then jumps again to $F_X(2) = \frac{1}{2}$ at $x = 2$, $F_X(3) = \frac{3}{4}$ at $x = 3$, and finally reaches its maximum value with final jump at $F_X(4) = 1$ at $x = 4$.

The cumulative distribution function F_X looks like a “step function” that “jump” by height 0.25 at each of the values 1, 2, 3, and 4. The height of each jump is proportional to the values of the probability mass function f_U assigns to each outcome.

Another way to obtain the graph of the function F_X is to use the helper function `plot_cdf` defined in module `plot_helpers`.

```
>>> plot_cdf(rvU, xlims=[0,10])
Output is shown in Figure 2.31.
```

code
2.3.21

To avoid repetition, we won’t show the plot-the-pmf and plot-the-CDF code examples for any of the other distribution discussed below, but you can copy-paste the above code examples and adapt them to use for other random variables.

Relations to other distributions

- The uniform distribution on with two outcomes $\mathcal{U}_d(0,1)$ is identical to the distribution $\text{Bernoulli}(\frac{1}{2})$.

See the links below for more about the discrete uniform distribution.

[Wikipedia article has more info and additional formulas]

https://en.wikipedia.org/wiki/Discrete_uniform_distribution

Bernoulli

The *Bernoulli* random variable $X \sim \text{Bernoulli}(p)$ describes a coin toss with a biased coin that has probability of heads p . The distribution is named after the mathematician Jacob Bernoulli, who did some important early work in probability theory.

The convention is to identify the outcomes of the coin toss as 0 for tails and 1 for heads. The sample space is therefore $\mathcal{X} = \{0, 1\}$. The probability mass function for the random variable $X \sim \text{Bernoulli}(p)$ is

$$f_X(x) = p^x(1-p)^{1-x}.$$

When $x = 0$, only the second factor remains: $f_X(0) = p^0(1-p)^{1-0} = (1-p)^1 = 1-p$. When $x = 1$, only the first term remains $f_X(1) = p^1(1-p)^{1-1} = p(1-p)^0 = p$.

If the coin is fair, then $p = 0.5$ and $1-p = 1-0.5 = 0.5$. We previously discussed the coin toss random variable C in several examples (see Section 2.1).

The mean and variance for a random variable $X \sim \text{Bernoulli}(p)$ are

$$\mu_X = \mathbb{E}[X] = p \quad \text{and} \quad \sigma_X^2 = p(1-p).$$

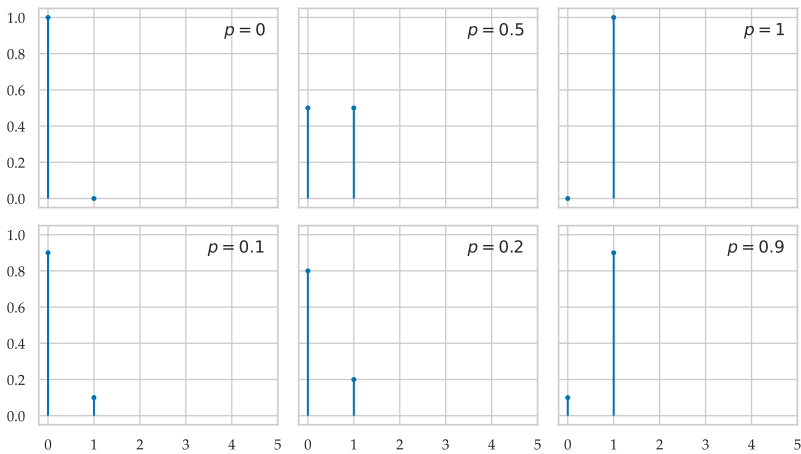


Figure 2.32: Plot of various Bernoulli distributions with various choices of the parameter p .

Computer model The code below shows how to create a Bernoulli random variable with parameter $p = 0.3$.

```
>>> from scipy.stats import bernoulli
>>> rvB = bernoulli(p=0.3)
```

code
2.3.22

The mean and variance of the random variable `rvB` are shown below.

```
>>> rvB.mean(), rvB.var()
(0.3, 0.21)
```

code
2.3.23

You can verify these are the same numbers as predicted by the formulas.

Calling the method `rvB.rvs(n)` will generate n random observations from the random variable `rvB`. Let's generate 10 observations to see how this works:

```
>>> rvB.rvs(10)
array([0, 1, 1, 0, 0, 0, 0, 1, 0, 1])
```

code
2.3.24

Note 4 out of the 10 outcomes were successes, which is close to the expected value $p = 0.3$.

Applications Many random phenomena besides coin tosses have two possible outcomes: success or failure, such as pass positive or negative outcome of a diagnostic test, converted vs non-converted for a website visitor, etc.

Relations to other distributions The Bernoulli distribution serve as building blocks for many of the other probability distributions.

- The distribution $\text{Bernoulli}(p = \frac{1}{2})$ is identical to the distribution $\mathcal{U}_d(0, 1)$
- The sum of n copies of a $\text{Bernoulli}(p)$ random variable is the binomial distribution $\text{Binom}(n, p)$ (defined below).
- The time-to-first-success in a series of independent $\text{Bernoulli}(p)$ observations is the *geometric* distribution (defined below).

Follow the links below to learn more about Bernoulli distributions.

[Bernoulli distribution on Wikipedia]

https://en.wikipedia.org/wiki/Bernoulli_distribution

Poisson

The Poisson distribution describes the number of random events that occur during some period of time, in a situation where each instant has the same probability of the event to occur.

We assume the events occur independently of each other at a constant average rate of λ . The probability mass function for the random variable $X \sim \text{Pois}(\lambda)$ is

$$f_X(x) = \frac{\lambda^x e^{-\lambda}}{x!},$$

where $x \in \{0, 1, 2, 3, \dots\}$. The parameter λ describes the average probability of the event during the chosen time period. Recall you've already seen this distribution when modelling the hard disk failures random variable H in Section 2.1.

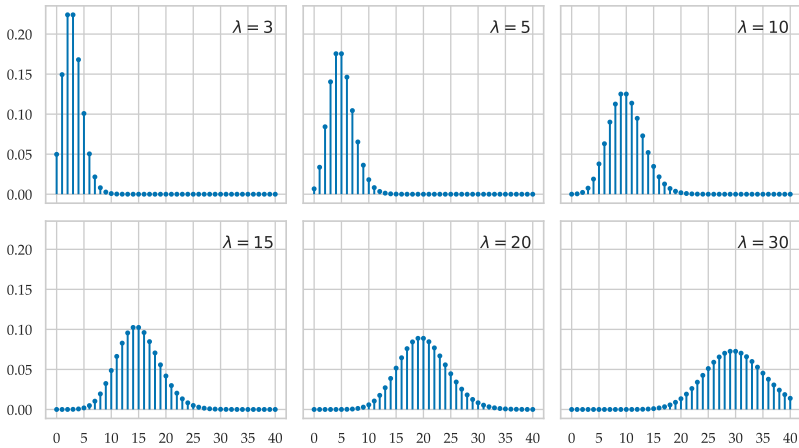


Figure 2.33: Probability mass function of the Poisson distribution for different values of the parameter λ .

Figure 2.33 shows the probability mass function of the Poisson distribution for different values of λ . As λ increases, the mean of the distributions shifts to larger values and also becomes more spread out.

The mean and variance for a random variable $X \sim \text{Pois}(\lambda)$ are

$$\mu_X = \mathbb{E}[X] = \lambda \quad \text{and} \quad \sigma_X^2 = \mathbb{E}_X[(X - \lambda)^2] = \lambda.$$

See E2.29 and P2.7 for the calculations.

Computer model To create the random variable object `rvP` equivalent to the random variable $X \sim \text{Pois}(\lambda = 10)$, you can use the following code:

```
>>> from scipy.stats import poisson
>>> lam = 10
>>> rvP = poisson(lam)
```

code
2.3.25

We used the variable name `lam` and not `lambda` because `lambda` is a reserved keyword in Python. (shorthand notation for defining functions; see Appendix C).

We can use the method `rvP.pmf(x)` to obtain values of the probability mass function $f_P(x)$, and the method `rvP.cdf(b)` to obtain values from the cumulative distribution function $F_P(b)$:

```
>>> rvP.pmf(8)
0.11259903214902009
>>> rvP.cdf(8)
0.3328196787507191
```

code
2.3.26

Applications The Poisson distribution is an important model for many phenomena we can observe in the real world:

- The number of phone calls a business will receive during the morning shift.
- The number of visitors to a website per minute.
- The number of customers arriving at a store.
- The number of radioactive decays...

The commonality in all these phenomena is that we're counting the total number of statistically independent events that occur at a constant rate λ (the expected number of the events per time period).

Relations to other distributions

- The Poisson distribution can be seen as a limiting case of the binomial distribution, if we take $\lambda = np$ and n goes to infinity. Expressed in math notation, this means $\lim_{n \rightarrow \infty} [\text{Pois}(np) = \text{Binom}(n, p)]$.
- The Poisson distribution is related to the negative binomial (Pascal) distribution: when $\lambda = n/p$ and $n \rightarrow \infty$, ... TODO
- Becomes the normal distribution for large value of λ , if we set $\sigma^2 = \lambda$ (and therefore (Ivan is assuming here) $\mu = \lambda$).
- The sum of two independent Poisson random variables is also a Poisson random variable. If $X_1 \sim \text{Pois}(\lambda_1)$ and $X_2 \sim \text{Pois}(\lambda_2)$, then $X_1 + X_2 \sim \text{Pois}(\lambda_1 + \lambda_2)$.

[Wikipedia page on the Poisson distribution]

https://en.wikipedia.org/wiki/Poisson_distribution

Binomial distribution

The binomial distribution models the number of successes in n consecutive draws from a Bernoulli distribution. Recall that the Bernoulli distribution with parameter p describes a coin toss of a biased coin with the probability of 1 (a.k.a. heads or “success”) is p , and therefore the probability of 0 (a.k.a. tails or “failure”) is $1 - p$. If we flip this coin n times, then the number of heads we’ll observe is described by the binomial random variable $X \sim \text{Binom}(n, p)$.

A random variable $X \sim \text{Binom}(n, p)$ has the probability mass function

$$f_X(x) = \binom{n}{x} p^x (1 - p)^{n-x}, \quad \text{for all } x \in \{0, 1, \dots, n\}.$$

The formula consists of the product of probabilities for x successes, $n - x$ failures, and the binomial coefficient $\binom{n}{x}$. This takes into account the number of ways the x successes can occur within the sequence of n trials. The name for this distribution comes from the binomial coefficient $\binom{n}{x}$ that appears in the formula.

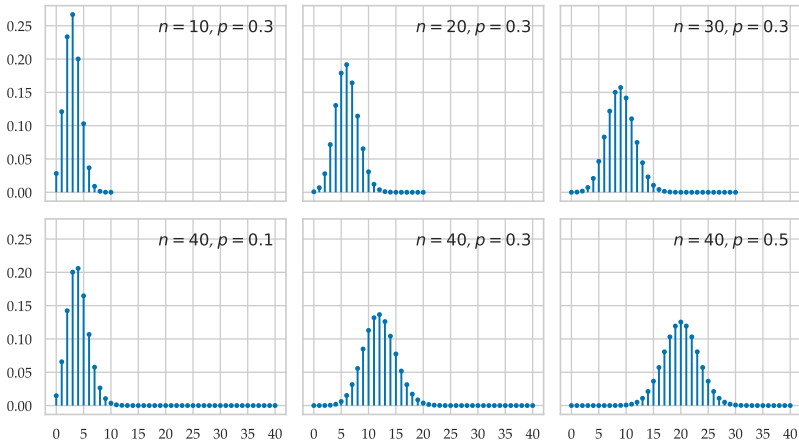


Figure 2.34: Plot of the probability mass function of the binomial distribution for different values of n and p .

Figure 2.34 shows the probability mass function for different values of p . Note the mean is np and the values get more spread out as p increases since $\sigma^2 = np(1 - p)$.

The mean and the variance of the distribution $\text{Binom}(n, p)$ are:

$$\mu_X = \mathbb{E}_X[X] = np \quad \text{and} \quad \sigma_X^2 = \mathbb{E}_X[(X - \mu_X)^2] = np(1 - p).$$

We can verify these formulas using the fact that the binomial distribution is equivalent to the sum of the outcomes of n independent variables B_1, B_2, \dots, B_n , where each $B_i \sim \text{Bernoulli}(p)$:

$$X = B_1 + B_2 + \dots + B_n,$$

Each B_i has mean p and variance $p(1-p)$, so when we combine them in a summation, we end up with the formulas $\mu_X = np$ and $\sigma_X^2 = np(1-p)$. In exercises E2.23 and E2.24, you'll be asked to verify the formula for μ_X . In problems P2.1 and P2.2, you'll be asked to derive the formula for σ_X^2 .

Computer model The code below creates a computer model for the binomial random variable for $n = 20$ observations with probability of success $p = 0.14$.

```
>>> from scipy.stats import binom
>>> n = 20
>>> p = 0.14
>>> rvX = binom(n,p)
```

code
2.3.27

The sample space of the random variable rvX is $\{0, 1, 2, \dots, 20\}$:

```
>>> rvX.support()
(0, 20)
```

code
2.3.28

The mean and the variance of rvX are given below.

```
>>> rvX.mean()
2.8
>>> rvX.var()
2.408
```

code
2.3.29

Applications The binomial distribution is very useful...

Example 1:

Example 2:

Relations to other distributions

- The Bernoulli distribution is a special case of the binomial distribution with $n = 1$.
- If $X \sim \text{Binom}(n, p)$ and $Y \sim \text{Binom}(m, p)$, then $X + Y \sim \text{Binom}(n + m, p)$.
- We can obtain the Poisson distribution from the binomial distribution by taking the limit $n \rightarrow \infty$ and $p \rightarrow 0$. See problem P2.3 for the details of this derivation.
- Normal approximation to the binomial: If the size of the sample n is large ($n \geq 20$), the normal distribution $X' \sim \mathcal{N}(\mu = np, \sigma = \sqrt{np(1-p)})$ can be used to approximate the binomial distribution $\text{Binom}(n, p)$. We'll learn more about this in Section 2.6 (see page 174).
- Consider a bucket that contains a "success" balls and b "failure" balls. The binomial distribution $\text{Binom}(n, p = \frac{a}{a+b})$ describes the number of successes can expect to observe if sample n balls from the bucket *with replacement*, meaning each ball is replaced back after being observed. If instead we select n balls from the bucket *without replacement*, then the number of successes is described by the hypergeometric distribution $\text{Hypergeom}(a, b, n)$, which we'll learn about later on in this section.

[The Wikipedia article on the binomial distribution]

https://en.wikipedia.org/wiki/Binomial_distribution

Geometric

The geometric distribution describes the distribution of the waiting time until the first success in a series of independent Bernoulli trials, where each Bernoulli trial has probability of a success p . The probability mass function of a random variable $X \sim \text{Geom}(p)$ is

$$f_X(x) = (1 - p)^{x-1} p,$$

where x is some positive integer. The formula consists of the product of $x - 1$ failure probabilities and once success. Note the sample space of the random variable X is $\mathcal{X} = \mathbb{N}^+ = \{1, 2, 3, \dots\}$, which is a countably infinite set. Indeed, there is no theoretical limit to the “bad luck” scenario in which the sequence of Bernoulli trials continues to result in failure. By definition, the trials must continue until the first success so the distribution is defined for all positive integers.

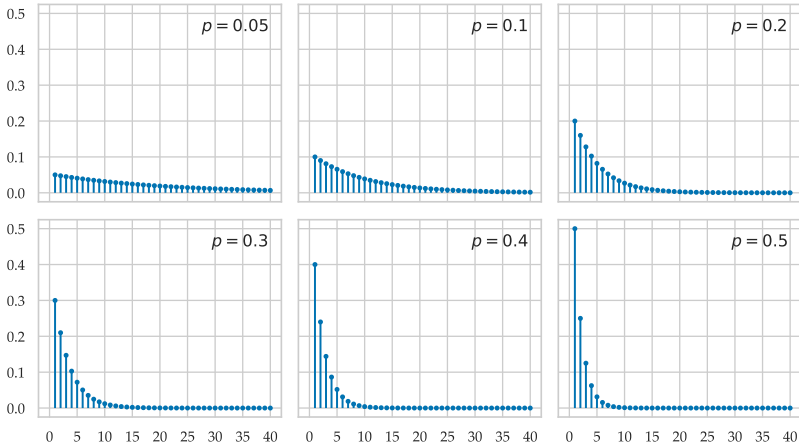


Figure 2.35: Plot of the probability mass function of the geometric distribution for different values of p . The higher the value of p , the more likely we are to observe the first success quickly.

The mean μ_X and the variance σ_X^2 of the random variable $X \sim \text{Geom}(p)$ are given by the formulas:

$$\mu_X = \mathbb{E}_X[X] = \frac{1}{p} \quad \text{and} \quad \sigma_X^2 = \mathbb{E}_X\left[\left(X - \frac{1}{p}\right)^2\right] = \frac{1-p}{p^2}.$$

See E2.25 and P2.4 for the derivations of these formulas.

The values of the probability mass function decrease geometrically by a factor of $r = (1 - p)$. Each subsequent trial, $f_X(x + 1) = (1 - p)f_X(x)$. This is where the name “geometric” comes from. Recall

the geometric sequence has the form $a_n = ar^n$, and its infinite series given by the formula $\sum_{n=0}^{\infty} ar^n = \frac{a}{1-r}$.

The geometric distribution has the memoryless property:

$$\Pr(X > s | X > t) = P(X > s - t).$$

This tells us ...

Computer model Let's create a computer model for a geometric distribution with parameter $p = 0.2$:

```
>>> from scipy.stats import geom
>>> rvG = geom(p = 0.2)
```

code
2.3.30

The sample space of the geometric distribution is $\{1, 2, 3, \dots\}$.

```
>>> rvG.support()
(1, inf)
```

code
2.3.31

The mean and variance of the random X are

```
>>> rvG.mean(), rvG.var()
(5.0, 20.0)
```

code
2.3.32

Applications We can model many situations with trials that are repeated until the first success occurs using the geometric distribution. For example, if the probability of success for some difficult task is p , then $f_X(x)$ represents the probability of succeeding on the x^{th} attempt. Persistence is the key, my friends! Alternatively, you can use the geometric distribution to compute the probability of failure after repeated successes. Suppose each time you turn on a light bulb it has a probability p of burning out, then you can $f_X(x)$ represent the probability of burning out on the x^{th} use, after $x - 1$ successes. In baseball, you can model the probability of a batter with average hit probability p of getting a hit on one of the first three attempts. In business, you could model the number of interviews you'll need to perform to hire a competent candidate as a geometric distribution, assuming each hiring interview has probability of success p .

Relations to other distributions

- If instead of stopping after the first success occurs, we continue counting until the first r successes occur, the waiting time will be described by the distribution $\text{NBinom}(r, p)$.

[Wikipedia article on the geometric distribution]

https://en.wikipedia.org/wiki/Geometric_distribution

Negative binomial (optional)

The geometric distribution describes repeated Bernoulli trials until the first success outcome. The *negative binomial distribution* is a generalization of a geometric distribution where we wait to obtain r successes. The probability mass function a random variable $X \sim \text{NBinom}(r, p)$ is

$$f_X(x) = \binom{x+r-1}{r-1} p^r (1-p)^x,$$

where p describes the probability of success, and x takes on values in the set $\{0, 1, 2, 3, \dots\}$. Note the random variable X counts the number of trials starting at the r^{th} , because we need to run at least r trials to obtain r successes. The last trial is necessarily a success, so the factor $\binom{x+r-1}{r-1}$ counts the number of ways of choosing the remaining $r-1$ successes among the $x+r-1$ trials before the last.

The name of this distribution comes from the fact that we can rewrite $\binom{x+r-1}{r-1}$ as an expression that involves the negative binomial coefficient $(-1)^x \binom{-r}{x}$. See E2.27 for the calculation. The negative binomial is sometimes called Pascal's distribution.

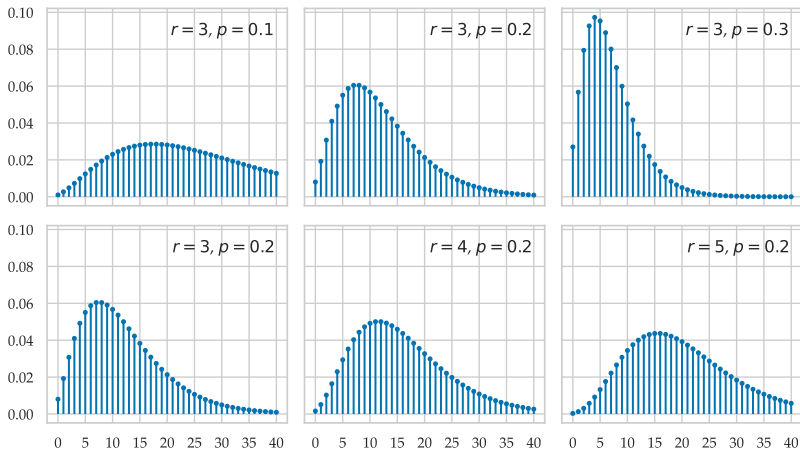


Figure 2.36: Plot of the probability mass function of the negative binomial distribution with different parameters.

The mean and variance of a random variable with negative binomial distribution $X \sim \text{NBinom}(r, p)$ are

$$\mu_X = \mathbb{E}[X] = \frac{r(1-p)}{p} \quad \text{and} \quad \sigma_X^2 = \mathbb{E}[(X - \mu_X)] = \frac{r(1-p)}{p^2}.$$

See E2.26 and P2.5 for the derivations.

Computer model Let's create the computer model rvN distributed according to $NBinom(r = 10, p = 0.2)$.

```
>>> from scipy.stats import nbinom
>>> r = 10
>>> p = 0.2
>>> rvN = nbinom(r, p)
```

code
2.3.33

The mean and the variance of the random variable rvN are calculated as follows.

```
>>> rvN.mean(), rvN.var()
(6.666666666666667, 11.111111111111111)
```

code
2.3.34

Verify these number answers by computing μ_X and σ_X^2 above.

Applications The negative binomial distribution comes up in many real-world situations where we're waiting for r independent events to occur. For example, consider a distributed storage in which information must be stored on at least $r = 3$ peers for redundancy. When a client wants to store the file f in the system, it needs to connect to different nodes of the storage system and attempt to save the file. The probability of completing the transfer to any node in the system is $p = 0.9$ (i.e. 10% error rate), so the client must repeatedly attempt the transfer to different nodes until it completes $r = 3$ transfers successfully. We can describe the probability of the client successfully saving the file after y attempts as a random variable $Y = r + X$, where $X \sim NBinom(r = 3, p = 0.9)$.

Relations to other distributions

- The negative binomial with $r = 1$ is the geometric distribution.
- The negative binomial is related to the sum of the geometric distribution: $NBinom(r, p) = \sum_{i=1}^r \text{Geom}(p)$.
- The negative binomial becomes the Poisson distribution for large r if we identify $\lambda = r(1 - p)$.
- The negative binomial can be derived as a gamma mixture of Poisson distributions. ????

[More details from the Wikipedia article]

https://en.wikipedia.org/wiki/Negative_binomial_distribution

Hypergeometric (optional)

Consider a bucket that contains a balls labelled “success” and b balls labelled “failure.” If we choose n balls randomly from this bucket, how many of the “success” balls will we observe?

In order to answer this question, we need to know the details of the sampling procedure. The probability distribution that describes the number of successes will be different depending on if we’re using sampling with or without replacement:

- *Sampling with replacement*: The process of choosing a ball from the bucket, recording its value (success or failure), then putting it back in the bucket.
- *Sampling without replacement*: the process of choosing a ball from the bucket, recording its value (success or failure), then putting it away.

When sampling is done *with replacement*, the probability of observing a “success” ball remains constant $p = \frac{a}{a+b}$ for all n draws, so the probability distribution that describes the number of successes in n draws is described by the binomial distribution $\text{Binom}(n, p = \frac{a}{a+b})$, which we’ve already discussed. See page 85.

When the sampling is done *without replacement*, the situation is much more complicated, since the probability of observing a success depends on the previous outcomes. For example, on the first draw, the probability of success will be $p_1 = \frac{a}{a+b}$, but on the second draw the probability of success can be either $p_2 = \frac{a}{a+b-1}$ or $p_2 = \frac{a-1}{a+b-1}$, depending on what we observed in the first draw. The probability of success on the third draw will in turn depend on the number of successes observed in the first two draws. and so on. The probability of observing a success on the k^{th} draw, will depend on what balls were observed on the previous $k - 1$ draws. Sounds complicated, right?

It turns out, mathematicians have thought about this process, and came up with a formula for the number of successes for the scenario of sampling with replacement: the hypergeometric distribution. The probability mass function of the random variable $X \sim \text{Hypergeom}(a, b, n)$ is

$$f_X(x) = \frac{\binom{a}{x} \binom{b}{n-x}}{\binom{a+b}{n}},$$

for $x \in \{0, 1, \dots, n\}$.

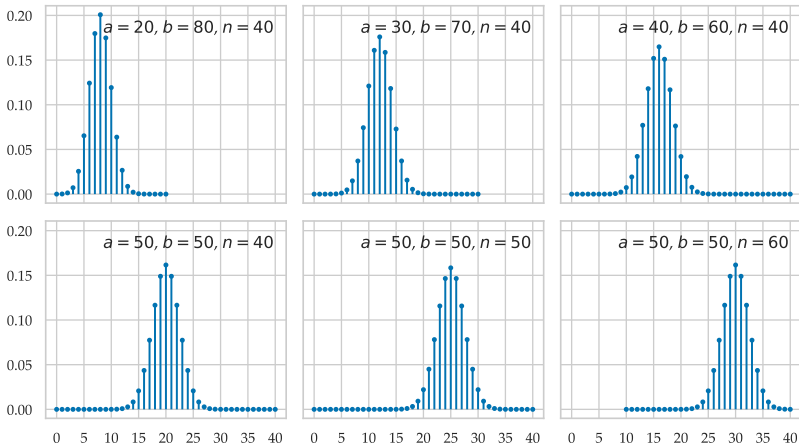


Figure 2.37: Plot of the probability mass function of the hypergeometric distribution with different parameters.

The mean and the variance of the random variable $X \sim \text{Hypergeom}(a, b, n)$ are

$$\mu_X = n \frac{a}{a+b} \quad \text{and} \quad \sigma_X^2 = n \frac{a}{a+b} \left(1 - \frac{a}{a+b}\right) \frac{a+b-n}{a+b-1}.$$

These formulas are easier to read if we define the quantity $p = \frac{a}{a+b}$, which corresponds to the probability of picking a success ball on the first draw:

$$\mu_X = np \quad \text{and} \quad \sigma_X^2 = np(1-p) \frac{a+b-n}{a+b-1}.$$

You'll be asked to derive the formula for the mean μ_X in E2.28, and the formula for the variance in P2.6.

Compare the equations for the mean and the variance of the hypergeometric with the equations for the mean and the variance of the Binomial distribution (see page 85). The resemblance is not a coincidence. Indeed, the binomial and hypergeometric distributions describe the same model, but the binomial distribution describes sampling with replacement while the hypergeometric distribution describes sampling without replacement. When x and $??$ are large numbers, the effect of sampling without replacement becomes negligible, and we can approximate the hypergeometric distribution using a binomial distribution. The factor $\frac{a+b-n}{a+b-1}$ is sometimes called the finite-sample size correction factor.

Computer model Let's use the hypergeom model from `scipy.stats` to create a random variable object `rvH` that corresponds to `Hypergeom(a = 30, b = 40, n = 20)`.

```
>>> from scipy.stats import hypergeom
>>> a = 30      # number of success balls
>>> b = 40      # number of failure balls
>>> n = 20      # how many we're drawing
>>> rvH = hypergeom(a+b, a, n)
```

code
2.3.35

Note we passed in the sum $a + b$ as the first argument when initializing the hypergeom model. This is because the function `hypergeom` expects the total number of balls as its first parameter.

The sample space of the random variable `rvH` is $\{0, 1, 2, 3, \dots, 20\}$.

```
>>> rvH.support()
(0, 20)
```

code
2.3.36

We can also compute the mean and the variance of the distribution using the appropriate methods.

```
>>> rvH.mean(), rvH.var()
(8.571, 3.549)
```

code
2.3.37

Applications Suppose you have a bag containing $a = 3$ good tomatoes and $b = 4$ rotten tomatoes. You want to choose $n = 2$ tomatoes from this bag to make a salad. What is the probability you will end up with zero, one, or two good tomatoes for your salad?

The situation is described by a hypergeometric random variable $X \sim \text{Hypergeom}(a = 3, b = 4, n = 2)$ with probability mass function:

$$f_X(x) = \frac{\binom{a}{x} \binom{b}{n-x}}{\binom{a+b}{n}} = \frac{\binom{3}{x} \binom{4}{2-x}}{\binom{7}{2}}.$$

Intuitively, the distribution counts the number of ways to choose x good tomatoes from the 3 good ones, times the number of ways to choose the remaining $2 - x$ from the 4 bad ones, divided by a normalization factor that describes all possible ways to choose any 2 tomatoes from a bag that contains a total of 7 tomatoes. In Exercise E2.32, you'll be asked you to obtain $f_X(0)$, $f_X(1)$, and $f_X(3)$ based on this formula.

We can also compute the probabilities of the three different outcomes, by thinking about the different sequences of observations that lead to each outcome. Since $n = 2$, we need to consider only two steps: the first draw and the second. The probability of picking zero good tomatoes is given by $f_X(0) = \frac{4}{7} \cdot \frac{3}{6} = 0.2857$, where $\frac{4}{7}$ is the probability of picking a bad tomato on the first draw, and $\frac{3}{6}$ is the probability of picking a bad tomato on the second draw. There

are two possible ways to pick one good tomato in two draws: the first one or the second one, so the probability of this outcome is $f_X(1) = \frac{4}{7} \cdot \frac{3}{6} + \frac{3}{7} \cdot \frac{4}{6} = 0.5714$. The probability of picking two good tomatoes is $f_X(2) = \frac{3}{7} \cdot \frac{2}{6} = 0.1429$.

Relations to other distributions

- If the draws from the are performed *with* replacements, the number of successes is described by the binomial distribution $\text{Binom}(p, n)$, where $p = \frac{a}{a+b}$ is the probability of drawing a success ball.
- Alternatively, if the number of balls a and b is very large, then sampling without replacement will be approximately the same as sampling with replacements, so the hypergeometric will resemble $\text{Binom}(\frac{a}{a+b}, n)$.

[Wikipedia article on the hypergeometric distribution]

https://en.wikipedia.org/wiki/Hypergeometric_distribution

2.3.5 Modelling real-world data

Let's try to connect probability distributions we discussed in this chapter with the data sets that we used as examples in the data chapters. The probability modelling skills you developed in this section, will help you model the data distributions in real-world datasets.

We'll plot the histogram of the data and a plot of the pmf in the same graph, then tweak the model parameters interactively until the two curves start to look the same.

Dataset 1: Website visitors conversion rates

Recall Alice's dataset of website visitors, which she collected to compare the current design (A) or the new design (B), is better at making visitors click the BUY NOW button. See page ?? for a reminder.

Each visitor to the website can be modelled as an instance of the Bernoulli random variable, where we identify $\text{bought}=1$ random variable with "success," and $\text{bought}=0$ as "failure."

If we sum together the number of conversions for the whole group, then

If we assume each Bernoulli trials are independent, then the total count of conversions is a binomial distribution. Since we have version A and version B of the website, we'll assume the probability of conversion is different for each website p_A and p_B . There were $n_A = 1014$ visitors assigned to the group A, and $n_B = 986$ visitors assigned to group B, so the two relevant distributions are:

$$X_A \sim \text{Binom}(n_A, p_A) \quad \text{and} \quad X_B \sim \text{Binom}(n_B, p_B).$$

Alice has observed the values $x_A = 47$ and $x_B = 56$ from these distributions (see Table ?? on page ??). She wants to know the value of the unknown parameters p_A and p_B , which correspond to the "conversion probabilities" for the two designs.

The process of "guessing" the model parameters based on data observations is called *estimation*, and this will be the main focus in the statistics chapter. Your knowledge of the properties of the binomial distribution will play an essential role.

Hard disk failure rates

Recall the example hard disk failure rate from page 25 in Section 2.1. In that example, we assumed the number of hard disks failures is distributed according to a Poisson distribution with parameter

$\lambda = 20$, $H \sim \text{Pois}(\lambda = 20)$, and used the computer model to obtain various estimates, and predictions for future observations.

2.3.6 Discussion

Summary of relations between distributions

TODO: FIGURE concept map showing the relations between prob. dists in this chapter

Figure 2.38

The relations shown in Figure 2.38 are only a subset of all possible relations that exist between probability distributions. For a more complete view see graph from [?] see <http://www.stat.rice.edu/~dobelman/courses/texts/leemis.distributions.2008amstat.pdf#page=3>.

Math formulas are optional

Even if this section had lots of math equations, I want you to remember that you can always use Python for practical calculations. If you need to compute $\binom{n}{k}$ (n choose k), you can use the math definition $\frac{n!}{(n-k)!k!}$, or call the Python function `comb(n,k)`.

Review of the `scipy.stats` methods

Recall the methods on discrete random variable objects, listed in Table 2.1 (page 75). These methods will come in handy when you try to solve the exercises.

Plotting distributions

The best way to understand a probability distribution is to plot its probability mass function. Recall the three-step procedure we showed in code block 2.3.17. 1. `arange` 2. `values of pmf` 3. `plt.stem` The function `plot_pmf` defined in `plot_helpers` module can perform these steps for you.

The procedure for plotting the CDF is slightly different, 1. `linspace` 2. `values of cdf` 3. `sns.lineplot` The function `plot_cdf` defined can perform these steps for you.

Plotting the probability functions is the best way to understand how their parameters change their shape. Try initializing random variables from different choices of parameters, and plot the distribution to see you get. Hands-on experience is always helpful after you have looked at equations.

Exercises

E2.23 Compute the mean of the random variable $X \sim \text{Binom}(n, p)$, using the interpretation that $X = B_1 + B_2 + \dots + B_n$, where each $B_i \sim \text{Bernoulli}(p)$ is drawn from the Bernoulli distribution with parameter p .

Hint: Recall the linear property of the expectation operator.

E2.24 Compute the mean of the random variable $X \sim \text{Binomial}(n, p)$ whose distribution is $p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$, for $k \in \{0, 1, \dots, n\}$.

E2.25 Compute the mean of the random variable $X \sim \text{Geometric}(p)$ with probability mass function $p_X(k) = (1-p)^{k-1} p$, for $k \in \{1, 2, \dots\}$.

Hint: You can use the formula $\sum_{k=1}^{\infty} k a r^{k-1} = -\frac{a}{(1-r)^2}$, which is obtained by taking the derivative of $\sum_{n=0}^{\infty} a r^n = \frac{a}{1-r}$ with respect to r .

E2.26 TODO: Replace $k \in r \dots$ to $x \in 0 \dots$;

Compute the mean of $X \sim \text{NegativeBinomial}(r, p)$ whose distribution is $p_X(k) = \binom{k-1}{r-1} (1-p)^{k-r} p^r$, for $k \in \{r, r+1, r+2, \dots\}$.

E2.27 TODO Replace $m = x$

Show that $\binom{k-1}{r-1}$ equals $(-1)^m \binom{-r}{m}$ where $m = k - r$.

Hint: Expand both expressions separately to show they are equal.

E2.28 Replace n with $a + b$

Replace K with a

Replace N with n

Find the mean of the random variable $X \sim \text{Hypergeometric}(n, K, N)$ with probability mass function $f_X(k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}}$, where k is between $\max\{0, n + K - N\}$ and $\min\{K, n\}$.

E2.29 Compute the mean of the random variable $X \sim \text{Poisson}(\lambda)$ with probability mass function $p_X(k) = \frac{(\lambda)^k e^{-\lambda}}{k!}$, for $k \in \{0, 1, 2, \dots\}$.

E2.30 Use the probability functions `dpois(k, lambda)`, `ppois(k, lambda)`, and `qpois(q, lambda)` to reproduce the calculations of the the number-of-hard-disk-failures scenario from page 2.1.5. Compute **a)** the probability of exactly 20, 21, and 22 failures, **b)** the probability of the event $\{16 \leq Z \leq 24\}$, and **c)** the 95 percentile $F_Z^{-1}(0.95)$ for the random variable $Z \sim \text{Poisson}(\lambda = 20)$.

E2.31 Use the probability functions in Excel to reproduce the calculations of the the number-of-hard-disk-failures scenario from page 2.1.5. Compute **a)** the probability of exactly 20, 21, and 22 failures, **b)** the probability of the event $\{16 \leq Z \leq 24\}$, and **c)** the 95 percentile $F_Z^{-1}(0.95)$ for the random variable $Z \sim \text{Poisson}(\lambda = 20)$.

E2.32 Create a random variable object from the `hypergom` computer model defined in `scipy.stats` that describes the scenario where we're choosing $n = 2$ tomatoes from a bag that contains $a = 3$ good tomatoes and $b = 4$ rotten tomatoes. Compute the probabilities $f_X(0)$, $f_X(1)$, and $f_X(2)$ using the computer model.

Hint: Recall that the `hypergom` takes the total number of items as the first argument.

E2.33 Amy is a veterinarian who helps all kinds of animals, but deep down inside she's a dog person, so she keeps always counts how many dog "patients" she sees each day. Suppose Amy's clinic received 20 pets today, of which 7 are dogs, and Amy will see will be able to see 12 of the 20 animals during her shift. What is the probability Amy will see exactly five dogs today?

Links

[Crash course in combinatorics]

<https://www.youtube.com/watch?v=ggNeQUe1Hj8>

[Graph showing the relations between probability distributions]

https://wikipedia.org/wiki/Relationships_among_probability_distributions

[Complete list of the discrete distributions available in SciPy]

<https://docs.scipy.org/doc/scipy/tutorial/stats/discrete.html>

[Description of probability distributions]

https://en.wikipedia.org/wiki/Probability_distribution

Congratulations on reaching the halfway point in the probability chapter. You now know everything you need to know about discrete distributions. In the next three sections, we'll learn about the probability theory with continuous distributions, which will first require introducing some math tools from calculus (integration). Logically, the probability theory with continuous random variables uses the same ideas as in the previous sections, but we're switching to a different math machinery for doing probability calculations.

This is a good moment to take a break from reading and try to solve some of the end-of-chapter problems (see page ZZ).

2.4 Calculus prerequisites

In order to do calculations with continuous random variables, you need to know about the calculus procedure of *integration*, which is used to compute the “total amount accumulated” of some quantity described by a continuous function, over an interval of inputs.

This section is your opportunity to learn (or relearn) the main concepts from calculus that you need to know to do calculations with continuous probability distributions. We’ll talk about concepts like sets, functions, and integrals, which are essential for understanding what’s going on in the rest of the book. We’ll start with a practical example in which integration is used. I want to show you that integration is not some fancy new idea, but an operation you’re already familiar with from everyday life.

Banking example Consider the function $\text{ba}(t)$ that represents your bank account balance at time t . Also consider the function $\text{tr}(t)$, which corresponds to the transactions (deposits and withdrawals) on your account.

Suppose you have a record of all the transactions on your account $\text{tr}(t)$, and you want to compute the final account balance at the end of the month $\text{ba}(30)$. You can use the integration procedure on the transactions $\text{tr}(t)$ to calculate the total change in the account balance at the end of the month, relative to the account balance at the beginning of the month $\text{ba}(0)$. The end-of-the-month-balance calculation is described by the following equation:

$$\text{ba}(30) = \text{ba}(0) + \int_0^{30} \text{tr}(t) dt.$$


The integral $\int_0^{30} \text{tr}(t) dt$ describes the process of computing the total of all the transactions that occurred between day 0 and day 30. The weird-looking integral sign “ \int ” comes from the Latin word *summa* for sum.

We use integrals every time we need to calculate the total of some quantity over a time period. The integral $\int_a^b q(t) dt$ is the calculation of the *total* of some quantity $q(t)$ that accumulates during the time period from $t = a$ to $t = b$.

2.4.1 Definitions

Let’s start by defining all the concepts from university-level math you need to know about. Don’t worry if you’re seeing some of these concepts for the first time, you’ll see plenty of examples using these

concepts, so you'll get to know them very well by the end of this section.

- *set*: a collection of math objects. Sets are denoted using curly brackets $\{\dots\}$. A set can be defined as a finite list of elements like $\{\text{heads}, \text{tails}\}$, by specifying a pattern $\{0, 1, 2, 3, \dots\}$, or through some other math expression $\{\langle \text{def}'n \rangle\}$.
- $f(x)$: a function of the form $f: \mathbb{R} \rightarrow \mathbb{R}$, which means f takes real numbers as inputs and produces real numbers as outputs. Functions are usually defined through an analytical formula like $f(x) = x^2$, which tells us how to compute the output $f(x)$ for a given input x . Functions can also be represented visually as a function graph , which is a curve that passes through all the coordinates pairs $(x, f(x))$ in the Cartesian plane.
- $A_f(a, b)$: the value of the *area* under the graph of the function $f(x)$ from $x = a$ until $x = b$. The area $A_f(a, b)$ corresponds to the following integral

$$A_f(a, b) \stackrel{\text{def}}{=} \int_a^b f(x) dx.$$

The \int sign stands for *sum*. Indeed, the integral is the “sum” of all the values of $f(x)$ for inputs x between $x = a$ and $x = b$.

- $F_0(b) \stackrel{\text{def}}{=} A_f(0, b)$: the *integral function* of $f(x)$. The integral function corresponds to the computation of the area under $f(x)$ as a function of the upper limit of integration:

$$F(b) \stackrel{\text{def}}{=} A_f(0, b) = \int_0^b f(x) dx.$$

The choice of $x = 0$ as the lower limit of integration is arbitrary. We could define any number of other integral functions $F_a(b)$ for different starting points $x = a$.

In the next few pages, we'll go into some details about each of these math concepts. Don't be intimidated by all the fancy-looking math notation—it's just a bunch of language mathematicians invented in order to describe concepts precisely and concisely. It looks weird to everyone who sees this specialized math notation for the first time (a.k.a. alien symbols), but you'll quickly get used to it.

2.4.2 Sets and intervals

Sets are arbitrary collections of math objects. Many math ideas are expressed using the language of sets, so it's worth going over the basic definitions and notation conventions.

- S, T : the usual variable names for sets
- $s \in S$: this statement is read “ s is an element of S ” or “ s is in S ”
- $\{ \text{definition} \}$: the curly brackets surround the definition of a set, and the expression inside the curly brackets describes what the set contains.
- S^c : the *complement* of the set S , is defined as all elements that are not in the set S .
- \mathbb{N} : the set of natural numbers $\mathbb{N} \stackrel{\text{def}}{=} \{0, 1, 2, \dots\}$
- \mathbb{Z} : the set of integers $\mathbb{Z} \stackrel{\text{def}}{=} \{\dots, -2, -1, 0, 1, 2, 3, \dots\}$
- \mathbb{Q} : the set of rational numbers, $\mathbb{Q} \stackrel{\text{def}}{=} \left\{ \frac{m}{n} \mid m \in \mathbb{Z}, n \in \mathbb{N}, n \neq 0 \right\}$.
The set \mathbb{Q} consists of all numbers that can be expressed as *fractions* of the form $\frac{m}{n}$, where m is an integer, n is a natural number, and $n \neq 0$.
- \mathbb{R} : the set of real numbers
- \mathbb{R}_+ : the set of nonnegative real numbers. The definition of the nonnegative is written as $\mathbb{R}_+ \stackrel{\text{def}}{=} \{ \text{all } x \text{ in } \mathbb{R} \text{ such that } x \geq 0 \}$, or it can be expressed more compactly as $\mathbb{R}_+ \stackrel{\text{def}}{=} \{ x \in \mathbb{R} \mid x \geq 0 \}$.

Note the multiple ways we use the curly-brackets notation $\{ \}$ to denote sets. A *finite set* is defined by simply listing all its elements. For example, the set of possible outcomes of a coin flip is $\{\text{heads}, \text{tails}\}$. For an infinite set we can't write down all the elements, but we can show the pattern like $\mathbb{N} \stackrel{\text{def}}{=} \{0, 1, 2, 3, 4, \dots\}$. The meaning of the three dots is “and so on, continuing the same pattern.” Another way to define a set is to use the *set-builder* notation $\{ \cdot \mid \cdot \}$. Inside the curly brackets we first describe the general kind of mathematical objects we are talking about, followed by the symbol “ \mid ” (read “such that”), followed by the conditions that must be satisfied by all elements of the set. The definitions of the rational numbers \mathbb{Q} and the nonnegative real numbers \mathbb{R}_+ above are examples of the set-builder notation.

The *number line* is a visual representation of the set of real numbers \mathbb{R} , as shown in Figure 2.39. The real numbers correspond to all the points on the number line, from $-\infty$ to ∞ .

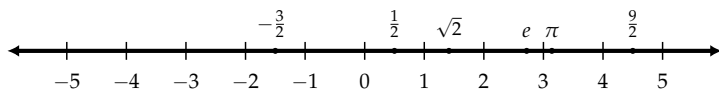


Figure 2.39: The real numbers \mathbb{R} cover the entire number line.

The set of real numbers includes all the rational numbers like $-\frac{3}{2}$, $\frac{1}{2}$, and $\frac{9}{2}$, as well as irrational numbers like $\sqrt{2}$, e , and π . This means

any number you are likely to run into when solving math problems can be visualized as a point on the number line.

Intervals

The number line can also be used to represent subsets of the real numbers, which we call *intervals*. Figure 2.40 shows an illustration of the interval $[2, 4] = \{x \in \mathbb{R} \mid 2 \leq x \leq 4\}$, which is a subset of the real numbers.

Here are some more examples of various intervals:

- $[a, b]$: the interval from a to b . This corresponds to the set of real numbers between a and b , including the endpoints a and b . The interval $[a, b]$ corresponds to the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$.
- $[a, \infty)$: the interval from a until infinity, which corresponds to the set $\{x \in \mathbb{R} \mid a \leq x\}$.
- $(-\infty, b]$: the interval from negative infinity until b , which corresponds to the set $\{x \in \mathbb{R} \mid x \leq b\}$.

The notation $[a, b]$ describes the *closed* interval from a to b , which means the endpoints a and b are included in the interval. The notation (a, b) describes the *open* interval from a to b , defined as the set $\{x \in \mathbb{R} \mid a < x < b\}$, which doesn't include the endpoints a and b . In other words, intervals defined using square brackets "[" include the endpoints (defined using less-than-or-equal conditions) while intervals defined with round brackets "(" do not include their endpoints (defined using strictly-less-than conditions). The distinction between open and closed intervals is important in general, but makes no difference in the context of probability theory, so you don't need to worry about the difference between $[a, b]$ and (a, b) in this book.

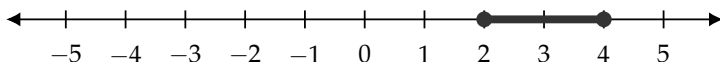


Figure 2.40: The interval $[2, 4] \stackrel{\text{def}}{=} \{x \in \mathbb{R} \mid 2 \leq x \leq 4\}$.

Set operations

We use set operations like union \cup , intersection \cap , and set difference \setminus to define composite sets.

- $S \cup T$: the *union* of two sets. The union of S and T corresponds to the elements in either S or T .
- $S \cap T$: the *intersection* of two sets. The intersection of S and T corresponds to the elements that are in both S and T .

- $S \setminus T$: *set difference* or *set minus*. The set difference $S \setminus T$ corresponds to the elements of S that are not in T .

Consider the overlapping intervals $A = [a, b]$ and $B = [c, d]$ illustrated in Figure 2.41. The union of these two intervals is the set of numbers that are *either* between a and b *or* between c and d , which corresponds to the interval $[a, d]$. The intersection of A and B is the set of numbers that are in *both* A and B , and corresponds to the interval $[c, b]$. The figure also illustrates the two set differences, $A \setminus B$ and $B \setminus A$ which correspond to numbers that are in one set, but not in the other.

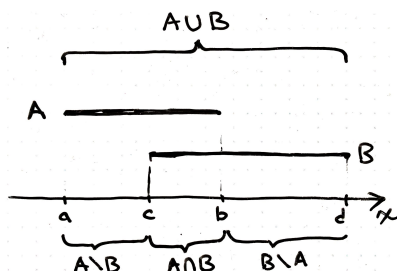


Figure 2.41: Various intervals that can be obtained using set operations of the intervals A and B .

I hope these definitions and examples made you feel more comfortable with sets, and the weird-looking curly bracket notation that mathematicians use to define sets. It might look a little complicated at first, but you'll get used to it in the rest of the book. In probability theory, we use finite sets and countably infinite sets like the natural numbers to represent the sample spaces of discrete random variables. We also use intervals to describe outcomes in the sample space of continuous random variables.

2.4.3 Functions

A *function* is a mathematical object that takes numbers as inputs and produces numbers as outputs. We use the notation

$$f: A \rightarrow B$$

to denote a function from the input set A to the output set B . For every input x , the output value of f for that input is denoted $f(x)$.

Function graph

The *graph* of a function is a line that passes through all input-output pairs of a function. Imagine we take out a piece of paper and

draw a coordinate system with a horizontal axis and a vertical axis. The horizontal axis describes the different input values x , while the vertical axis describes the output values $f(x)$. Each input-output pair of the function f corresponds to the point $(x, f(x))$ in the coordinate system. We obtain the graph of the function by varying the input coordinate x and plotting all the points $(x, f(x))$, as illustrated in Figure 2.42.

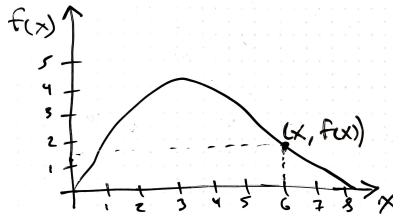


Figure 2.42: The graph of the function f consists of all the points with coordinates $(x, f(x))$ over some interval of x values.

The graph of the function f allows us to see at a glance the behaviour of the function for all possible inputs, and forms an essential visualization tool. Indeed, many phenomena and calculations related to functions can be understood geometrically as operations based on the graph of the function.

In probability theory, we use functions to describe the probability distributions of random variables. Discrete random variables are described by a probability mass function of the form $f: \mathcal{X} \rightarrow \mathbb{R}$, where the sample space \mathcal{X} is either a finite set or a countably infinite set like the natural numbers \mathbb{N} . Continuous random variables are described by probability density functions of the form $f: \mathcal{X} \rightarrow \mathbb{R}$, where the sample space \mathcal{X} is some subset of the real numbers \mathbb{R} .

Python functions

In Python, functions are defined using the `def` keyword. For example, the code below shows how to define the function $g(x) = x^2$.

```
>>> def g(x):
    return x**2
```

code
2.4.1

The first line specifies we're defining a function called `g` that takes the variable `x` as input (function inputs are also called *arguments*). The function body can contain multiple lines and compute arbitrary intermediate values. The `return` statement (usually the last line in the function body) specifies the output value of the function.

We *call* a Python functions using its name followed by the input value in brackets, which is identical to the math notation.

```
>>> g(4)
16
```

Consult the Python tutorial in Appendix C for more details about the syntax for defining functions.

Plotting functions in Python

An easy way to plot the graph of the function $g(x)$ is to use the `numpy` and `seaborn` modules, as shown in the code example below.

```
>>> import numpy as np
>>> import seaborn as sns
>>> xs = np.linspace(0, 10, 100)
>>> gxs = g(xs)
>>> sns.lineplot(x=xs, y=gs, label="Graph of g(x)")
See Figure 2.43 for the output.
```

code
2.4.3

The first two lines import the modules `numpy` and `seaborn` under the aliases `np` and `sns`. We use the function `np.linspace` to create an array (a list of numbers) `xs`, which contains 100 input values that range from $x = 0$ until $x = 10$. Next we apply the function `g` to the array of inputs `xs` and store the result in the array `gs`, which contains all the output values of the function for the input values `xs`. At this point, the arrays `xs` and `gs` contain 100 input-output pairs of the form $(x, g(x))$, which is exactly what we need to plot the graph of the function. On the last line, we call the function `lineplot` to create the graph of $g(x)$, which produces the plot shown in Figure 2.43.

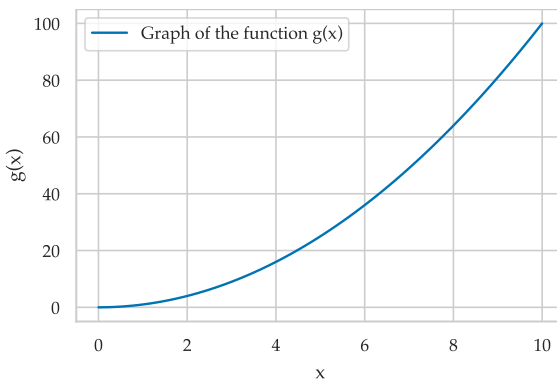


Figure 2.43: Graph of the function $g(x) = x$ from $x = 0$ until $x = 10$.

Note the steps we used to obtain the function graph in code 2.4.3 correspond exactly to the mathematical procedure for drawing the graph of $f(x)$: draw the line that passes through all $(x, f(x))$ input-output pairs.

Using pre-defined Python functions

For many math calculations, we can reuse pre-defined functions that are available in Python modules like NumPy, SciPy, and SymPy. For example, if we need to calculate the square root of 4 or the natural logarithm of 4, we can import the Python module `numpy` and call the functions `sqrt` and `log` defined in that module.

```
>>> import numpy as np
>>> np.sqrt(4)
2.0
>>> np.log(4)
1.3862943611198906
```

code
2.4.4

Inverse functions

The inverse function $f^{-1}: B \rightarrow A$ performs the *inverse operation* of the function $f: A \rightarrow B$. If you start from some x , apply f , and then apply f^{-1} , you'll arrive—full circle—back to the original input x :

$$f^{-1}(f(x)) = x.$$

In Figure 2.44 the function f is represented as a forward arrow, and the inverse function f^{-1} is represented as a backward arrow that puts the value $f(x)$ back to the x it came from.

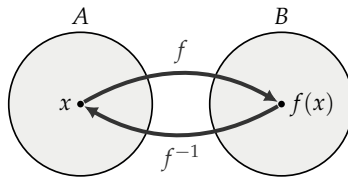


Figure 2.44: The inverse f^{-1} undoes the operation of the function f .

For example, if we compute the square root of a number, then square the result, we obtain the original number, since the quadratic function x^2 is the inverse of the square-root function \sqrt{x} .

```
>>> np.sqrt(4)**2
4.0
```

code
2.4.5

The exponential function e^x is the inverse of the logarithmic function $\log_e(x)$, so if we compute the logarithm of a number then apply the exponential function, we get back the original input.

```
>>> np.exp(np.log(4))
4.0
```

code
2.4.6

In probability theory, we often do calculations using the cumulative distribution function (CDF) $F_X: \mathcal{X} \rightarrow [0, 1]$, and also use the inverse of the cumulative distribution function $F_X^{-1}: [0, 1] \rightarrow \mathcal{X}$. Knowing about inverse functions (and the weird superscript $^{-1}$ notation used to describe them) is useful for your conceptual understanding of these concepts: instead of thinking about the inverse-CDF F_X^{-1} as some new complicated concept you have to memorize, you can think of F_X^{-1} as the “undo operation” for F_X . In other words, F_X and F_X^{-1} describe the same mapping, but used in opposite directions.

2.4.4 Integrals as area calculations

An integral corresponds to the computation of the *area* enclosed between the curve $f(x)$ and the x -axis over some interval of x values:

$$A_f(a, b) = \int_{x=a}^{x=b} f(x) dx.$$

We refer to the numbers a and b as the *limits of integration*, and the notation $\int_a^b f(x) dx$ is shorthand for $\int_{x=a}^{x=b} f(x) dx$.

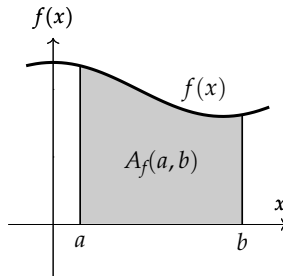


Figure 2.45: The integral of the function $f(x)$ between $x = a$ and $x = b$ corresponds to the shaded area.

The notion of an integral is foundational for understanding continuous random variables. Every time we compute the probability of some outcome of a continuous random variable, there is an integral calculation going on under the hood, so integrals is not a topic you can skip.

If this is the first time you’re learning about integrals, it’s understandable if you feel intimidated by the complicated math notation, but you have to trust me on this one: except for the notation, there is nothing to worry about! In the next few pages, I’ll do my best to introduce you to the topic of integrals, and you’ll learn three different ways to do compute integrals.

Let’s start with some examples.

Example 1: integral of a constant function

Consider the constant function $f(x) = 3$. No matter what the input x is, the output is always 3. We can easily find the area under the graph of the function $f(x)$ between any two points, since the region under the graph has a rectangular shape. See Figure 2.46 for an illustration.

The area under $f(x)$ between $x = 0$ and $x = 5$ corresponds to the following calculation:

$$A_f(0, 5) = \int_0^5 f(x) \, dx = 3 \cdot 5 = 15.$$

The area under the graph of $f(x)$ is a rectangle with height 3 and width 5, so its area is $3 \cdot 5 = 15$.

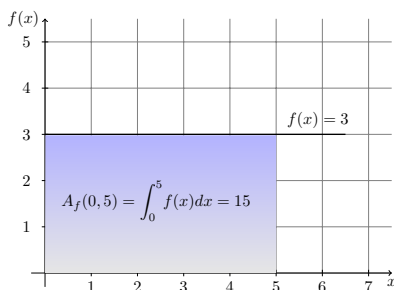


Figure 2.46: The area of a rectangle of height 3 and width 5 is equal to 15.

Example 2: integral of a linear function

Consider now the area under the graph of the line $g(x) = x$ between $x = 0$ and $x = 5$, as shown in Figure 2.47. Since the region under the curve is triangular, we can compute its area using the formula for the area of a triangle, which is “base times height divided by 2.”

The integral of $g(x)$ from $x = 0$ until $x = 5$ is described by the following calculation:

$$A_g(0, 5) = \int_0^5 g(x) \, dx = \frac{1}{2} 5 \cdot 5 = \frac{1}{2} 5^2 = \frac{25}{2} = 12.5.$$

I hope these examples helped you see that the scary-looking integral sign is not that complicated after all. It’s just a fancy way to describe “area under the graph of a function” calculations.

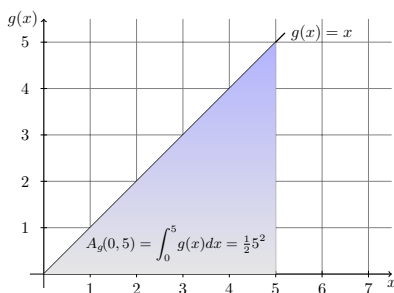


Figure 2.47: The area of a triangle with base 5 and height 5 is equal to $\frac{1}{2}5^2 = \frac{25}{2} = 12.5$.

Properties of integrals

We'll now state some properties of integrals that follow from their interpretation as area calculations.

- **Additivity.** The integral from a to b plus the integral from b to c is equal to the integral from a to c :

$$\int_a^b f(x) dx + \int_b^c f(x) dx = \int_a^c f(x) dx.$$

- **Constant multiple of a function.** The integral of the function $cf(x)$ is equal to c times the integral of $f(x)$, for any constant c :

$$\int cf(x) dx = c \int f(x) dx.$$

- **Sum of two functions.** The integral of a sum of two functions is equal to the sum of the integrals of the individual functions:

$$\int [f(x) + g(x)] dx = \int f(x) dx + \int g(x) dx.$$

- **Linearity.** The combination of the above two properties tells us that integration is a *linear* operation, meaning it preserves linear combinations. The integral of the linear combination of two functions $\alpha f(x) + \beta g(x)$, is equal to the same linear combination of the integrals of the two functions:

$$\int [\alpha f(x) + \beta g(x)] dx = \alpha \int f(x) dx + \beta \int g(x) dx,$$

where α and β are two arbitrary constants.

- **Integral at a single point.** Integrals over intervals with zero length have zero value for any function $f(x)$:

$$\int_a^a f(x) dx = 0.$$

Thinking geometrically, this integral defines a region with height $f(x)$ and width 0, so it corresponds to zero area.

2.4.5 Integrals as functions

The *integral function* $F_0(b)$ corresponds to the area calculation with a variable upper limit of integration $A_f(0, b)$. The variable b , which serves as the input for the integral function F_0 , corresponds to the upper limit of integration in the following calculation:

$$F_0(b) \stackrel{\text{def}}{=} A_f(0, b) = \int_{x=0}^{x=b} f(x) dx.$$

There are two variables and one constant in this formula. The input variable b describes the upper limit of integration. The *integration variable* x performs a sweep from $x = 0$ until $x = b$. The constant 0 describes the lower limit of integration. As a matter of convention, we'll always denote the integral function using the capital letter of the same letter as the original function.

Note that choosing $x = 0$ for the starting point of the integral function was an arbitrary choice, and we obtain another integral function if we use $x = a$ as the starting point, $F_a(b) = \int_a^b f(x) dx$. Two integral functions differ only by a constant term. For example, $F_0(b) = F_a(b) + C$, where $C = \int_{x=0}^{x=a} f(x) dx$.

The integral function $F(b)$ contains the “precomputed” information about the area under the graph of $f(x)$. The area under $f(x)$ between $x = a$ and $x = b$ can be obtained by calculating the *change* in the integral function as follows:

$$A_f(a, b) = \int_a^b f(x) dx = F(b) - F(a).$$

Intuitively, this formula computes the area $A_f(a, b)$ as the difference between the areas of two regions: the area until $x = b$ minus the area until $x = a$, as illustrated in Figure 2.48.

Example 1 revisited

We can easily find the integral function for the constant function $f(x) = 3$, because the region under the curve is rectangular.

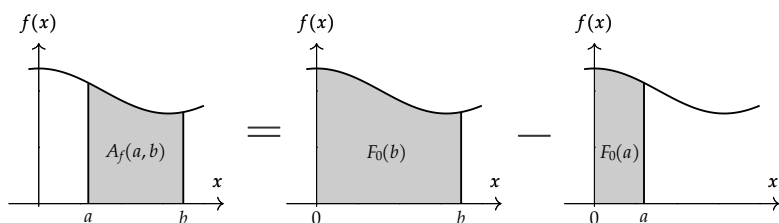


Figure 2.48: The area under $f(x)$ between $x = a$ and $x = b$ is computed using the formula $A_f(a, b) = F_0(b) - F_0(a)$, which describes the change in the output of $F_0(x)$ between $x = a$ and $x = b$.

Choosing $x = 0$ as the starting point, we obtain the integral function $F_0(b)$ that corresponds to the area under $f(x)$ between $x = 0$ and $x = b$ as follows:

$$F_0(b) = A_f(0, b) = \int_0^b f(x) dx = 3b.$$

The area is equal to the rectangle's height times its width, as illustrated in Figure 2.49.

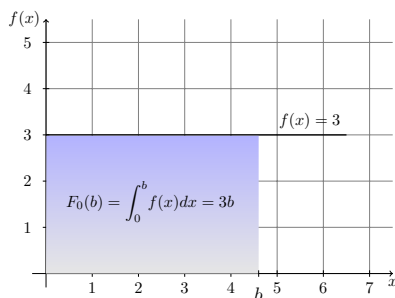


Figure 2.49: The area of a rectangle of height 3 and width b is equal to $3b$.

Knowing the function $F_0(b)$ allows us to compute the area under the graph of $f(x)$ between any two points $x = a$ and $x = b$ using the formula $A_f(a, b) = F_0(b) - F_0(a) = 3(b - a)$.

Example 2 revisited

Consider now the area under the graph of the line $g(x) = x$, starting from $x = 0$. Since the region under the curve is triangular, we can compute its area using the formula for the area of a triangle: base times height divided by two.

The general formula for the area under $g(x)$ from $x = 0$ until $x = b$ is described by the following integral calculation:

$$G_0(b) = A_g(0, b) = \int_0^b g(x) dx = \frac{1}{2}(b \times b) = \frac{1}{2}b^2.$$

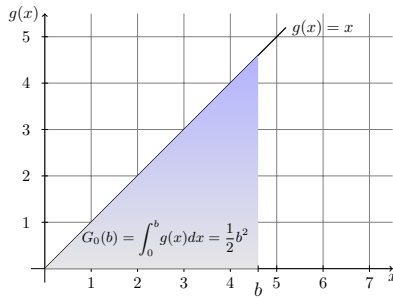


Figure 2.50: The area of a triangle with base b and height b is equal to $\frac{1}{2}b^2$.

Knowing the function $G_0(b)$ allows us to compute the area under the graph of $g(x)$ between $x = a$ and $x = b$ as the difference $A_g(a, b) = G_0(b) - G_0(a) = \frac{1}{2}b^2 - \frac{1}{2}a^2$.

We were able to compute the above integrals thanks to the simple geometries of the areas under the graphs. Computing integrals of more complicated functions requires more advanced techniques. There is an entire course called integral calculus which is dedicated to the task of finding integrals using various tricks and techniques.

Taking a calculus course would be useful if you plan to study physics or engineering, but for the purpose of learning probability and statistics, you're not required to learn all these integration techniques. Instead, you can rely on computers to do integration for you. Specifically, you can use the Python modules SciPy and SymPy to compute all the integrals you need, as we'll show in the next two sections.

2.4.6 Computing integrals numerically using SciPy

There are numerous ways to compute integrals using Python. Computing integrals “numerically” means we're splitting the region of integration into thousands or millions of subregions, computing the areas of these subregions, then adding up the areas of the subregions to obtain the total area.

The Python function `quad` in the module `scipy.integrate` allows us to compute the integral of any function. The name `quad`

is short for “quadrature” which is the historical math term used for find-the-area procedures. Let’s start by importing the quad function.

```
>>> from scipy.integrate import quad
```

code
2.4.7

Now let’s define a Python function f that corresponds to the constant function $f(x) = 3$.

```
>>> def f(x):
        return 3
>>> f(333)
3
```

code
2.4.8

No matter what input x we choose, the output will always be the same $f(x) = 3$.

To compute the integral $\int_0^5 f(x)dx$ we call the function quad with inputs f as the first argument, and the limits of integration $a = 0$ and $b = 5$ as the second and third arguments.

```
>>> quad(f, 0, 5)
(15.0, 1.1102230246251565e-13)
```

code
2.4.9

The function quad returns a tuple (a pair of numbers) as output: (A, ϵ) . The first number in the tuple is the value of the area calculation. The second number ϵ tells us the accuracy of the procedure used to calculate the area. In the above calculation, the output tells us the integral $\int_0^5 f(x)dx$ is equal to 15.0 up to a precision on the order of 10^{-13} .

Since we’re usually only interested in the value of the area A and not the precision ϵ , we often select the first number in the output of quad. This is why you’ll often see the expression `quad(...)[0]` in code examples.

```
>>> quad(f, 0, 5)[0] # extract A
15.0
```

code
2.4.10

As a second example, let’s calculate the area under the graph of the function $g(x) = x$ between $x = 0$ and $x = 5$.

```
>>> def g(x):
        return x
>>> quad(g, 0, 5)[0]
12.5
```

code
2.4.11

The answer we obtained matches the results of the general formula we obtained above, $A_g(0,5) = \frac{1}{2}b^2$, when the upper limit of integration is $b = 5$.

We’ll use the function quad hundreds of times in the remainder of the book to compute various integrals as part of probability and statistics calculations, so make sure you understand what is going on in the above code examples. The main takeaway message is that the quad function is your friend whenever you need to compute

integrals. All the scary-looking math equations that contain the \int symbol can be computed using one or two lines of Python code.

Trapezoid approximation

There is another way to compute the area under the graph of $f(x)$ called the *trapezoid approximation*, which I want you to know about. Note, however, this section is optional reading material and you can totally skip it if you prefer.

In mathematics, a *trapezoid* is a quadrilateral (a geometric figure with four sides) that has two sides that are parallel. We can compute the integral $\int_a^b f(x)dx$ by splitting up the region under the graph of $f(x)$ into a number of smaller trapezoid-shaped subregions. Figure 2.51 shows an illustration of the trapezoid approximation calculation under the graph of some function $f(x)$ using $n = 10$ subregions. The combined area of the 10 trapezoids is an approximation of the area of the region under the graph of $f(x)$.

Notice the first trapezoid (actually a triangle) is an overestimate for the area under the graph of $f(x)$, since its area is greater than the area under the graph of $f(x)$. In contrast, the second and third subregions are underestimates, since they don't cover all the area under the function. This issue of underestimation and overestimation can be addressed by increasing the number of subregions. If we split the region into $n = 100$ trapezoids, the size of the under- and over-estimates will become much smaller, and a split into $n = 1000$ subregions will produce an even more accurate approximation. The illustration with $n = 10$ subregions allows us to see how the trapezoid approximation works, but for practical calculations we always compute trapezoid approximations using thousands or tens of thousands of subregions to get more accurate results.

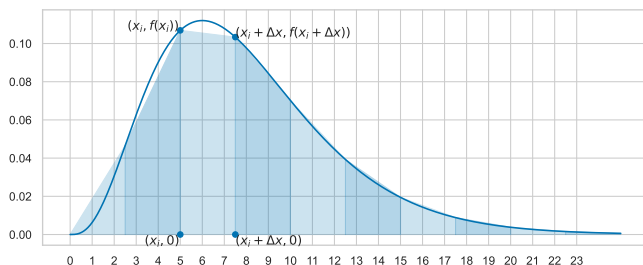


Figure 2.51: Illustration of the trapezoid approximation to the integral $\int_0^{25} f(x)dx$ using $n = 10$ trapezoids.

Figure 2.51 illustrates the calculation of the area under $f(x)$ by approximating it using $n = 10$ trapezoid subregions. We've

highlighted the corners of the third trapezoid subregion, to give you some intuition about the data that is required to compute this approximation. The integral we want to compute is from $a = 0$ until $b = 25$. We split the region into $n = 10$ subregions, so the width of each subregion is $\Delta x = \frac{b-a}{n} = \frac{25-0}{10} = 2.5$. The x -coordinate of the regions are given by $xs = [0, 2.5, 5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25]$, and we also need to know the value of the function $f(x)$ for all the coordinates x in the list xs .

We don't need to know the actual formula for the area of a trapezoid, since the computer will compute it for us. We just need to give it the lists containing the x -values and $f(x)$ -values, as you'll see in the code examples below.

Let's use the trapezoid approximation to compute the integral $\int_0^5 g(x)dx$, where $g(x) = x$. We already computed this integral two times earlier using the simple geometry of the region and using the quad function, and we found out the area under $g(x)$ over the interval $[0, 5]$ is 12.5, so we better be getting the same answer!

```
>>> from scipy.integrate import trapz
>>> n = 1000
>>> xs = np.linspace(0, 5, n)
>>> gxs = g(xs)
>>> trapz(gxs, xs)
12.5
```

code
2.4.12

On the first line, we import the function `trapz` from the module `scipy.integrate`. On the second line, we use the function `np.linspace` to create an array `xs` (a list of numbers) that contains $n = 1000$ input values x that go from $x = 0$ until $x = 5$. On the third line, we compute the value of the function $g(x)$ for each of the input values, and store the result in another array called `gx`s. Finally, we call the function `trapz(gxs, xs)` with the two arrays to compute the trapezoid approximation.

2.4.7 Computing integrals symbolically with SymPy

We can also use Python to do *symbolic* integration using variables (symbols) instead of numbers. Symbolic integration allows us to obtain exact formulas for integrals that are valid for *any* limits of integration $x = a$ and $x = b$. The Python module `sympy` provides the functionality for doing symbolic math calculations similar to the calculation you could do using pen and paper.

The following code block imports the SymPy function `symbols`, which is used to define new symbolic variables, and the function `integrate` that we'll use for computing integrals.

```
>>> from sympy import symbols, integrate
```

code
2.4.13

Next we define four symbols x , a , b , and c , which we'll use to denote mathematical variables and constants in the following code examples.

```
>>> x, a, b, c = symbols('x a b c')
```

code
2.4.14

Example 1 revisited again

Consider the constant function $f(x) = c$. The symbolic expression that represents the value of this function is simply the constant c , which we can define as follows:

```
>>> fx = c
>>> fx
c
```

code
2.4.15

The variable `fx` is defined as the constant c , one of the SymPy symbols we defined earlier, which we assume corresponds to some unspecified constant value.

To compute the integral $\int_a^b f(x)dx$, we call the SymPy function `integrate`, passing in the expression we want to integrate as the first argument. The second argument is a triple (x, a, b) , which specifies the variable of integration x , the lower limit of integration a , and the upper limit of integration b .

```
>>> integrate(fx, (x,a,b)) # = A_f(a,b)
c*(b-a)
```

code
2.4.16

Since a , b , and c are arbitrary constants, the expression we obtain for $A_f(a, b) = \int_a^b f(x)dx$ is a general purpose formula that works for all functions $f(x) = c$ and all possible integration intervals $[a, b]$. Geometrically speaking, this is just the height-times-width formula for the area of a rectangle.

To compute the specific integral between $a = 0$ and $b = 5$ under the graph of $f(x) = 3$, we can use the method `subs` (short for substitute) on the SymPy expression we obtained as a result of the integration. The `subs` method expects as inputs a Python dictionary whose keys are symbols, and whose values represent the numbers we want to “plug” into the expression. In our case, we want to make the substitutions $c = 3$, $a = 0$, and $b = 5$.

```
>>> integrate(fx, (x,a,b)).subs({c:3, a:0, b:5})
15
```

code
2.4.17

This result matches the value we obtained using the intuitive geometrical calculation (see Figure 2.46) and the value we obtained using numerical integration, `quad(f, 0, 5) = 15`.

We can also use SymPy to compute the integral function $F_0(b)$, which is defined as $F_0(b) \stackrel{\text{def}}{=} \int_0^b f(x)dx$, for the function $f(x) = fx$.

code
2.4.18

```
>>> integrate(fx, (x,0,b)) # = F_0(b)
b*c
```

Recall that the integral function F_0 is simply the area-under-the-graph calculation with a variable upper limit of integration b . See Figure 2.49 for an illustration of the integral function $F_0(b)$.

Example 2 revisited again

Let's now compute some integrals of the function $g(x) = x$. First we'll define a SymPy expression that corresponds to the function.

```
>>> gx = 1*x
>>> gx
x
```

code
2.4.19

We can now compute the integral $\int_a^b g(x)dx$ by calling the function `integrate` with arguments `gx`, followed by the triple specifying the variable of the integration and the limits of integration.

```
>>> integrate(gx, (x,a,b)) # = A_g(a,b)
b**2/2 - a**2/2
```

code
2.4.20

To obtain the numerical value for the integral $\int_0^5 g(x)dx$, we call the method `subs` on the result of the integration.

```
>>> integrate(gx, (x,a,b)).subs({a:0, b:5})
25/2
```

code
2.4.21

SymPy computed the exact answer for us as a fraction $\frac{25}{2}$, but we sometimes want to force the answer to be computed as a floating-point number (a Python `float`), which we can do by calling the `.evalf()` method on the SymPy expression.

```
>>> integrate(gx, (x,a,b)).subs({a:0, b:5}).evalf()
12.5
```

code
2.4.22

This result matches the value we obtained earlier using numerical integration, $\text{quad}(g,0,5) = 12.5$.

If we use the symbol b for the upper limit of integration, we can obtain an expression for the integral function $G_0(b) \stackrel{\text{def}}{=} \int_0^b g(x)dx$.

```
>>> integrate(gx, (x,0,b)) # = G_0(b)
b**2 / 2
```

code
2.4.23

Note the expression for $G_0(b)$ we obtain from SymPy is identical to the formula we obtained earlier using a geometrical calculation (the area of a triangle with base b and height b). See Figure 2.50.

Unfortunately, it's not always possible to use symbolic manipulations to find integrals. We can only use `sympy.integrate` for certain simple examples where it is possible to obtain exact expressions

for integral functions. For most practical calculations in probability and statistics, we'll need to rely on the `scipy.integrate` function `quad(f, a, b)`, which computes the integral $\int_a^b f(x)dx$ for *any* function $f(x)$ expressed as a Python function `f`.

2.4.8 Other calculus topics

In this section, we focused on integration, which is the main tool you need to “import” from calculus to understand probability theory. There are many other interesting calculus topics like: limits, derivatives, sequences, and series. Learning calculus will introduce you to a number of mind-expanding theoretical results, like the *fundamental theorem of calculus*, for example.

We'll now give a bird's-eye view of these other topics in calculus, and provide links to books and other resources you can use to learn more about calculus.

Limits

In high school math, we learn all kinds of math procedures for solving problems using a finite number of steps of math operations. Whether you're manipulating expressions using algebra, or applying the inverse function to simplify an equation, all problems in high school math can be solved by using less than five steps, or if your teacher really doesn't like you 10 steps. In calculus, we learn a broader class of problem-solving strategies that include procedures with an infinite number of steps.

Limit expressions provide a precise mathematical language for talking about infinitely large numbers, infinitely small steps, and mathematical procedures with an infinite number of steps. Here are three representative examples of limit expressions:

- $\lim_{x \rightarrow \infty} f(x)$: limit expression that describes what happens to $f(x)$ when the input to the function x tends to infinity (gets larger and larger). In words, this limit expression is read as “limit of $f(x)$ as n goes to infinity.”
- $\lim_{n \rightarrow \infty} g(n)$: limit expression that describes the value of $g(n)$ as the integer n tends to infinity. The integer n usually describes the number of steps in a given procedure, and $g(n)$ describes the output of this procedure when n steps are used.
- $\lim_{\delta \rightarrow 0} h(\delta)$: limit expression that describes what happens to $h(\delta)$ as the real number δ tends to zero. The number δ (the Greek letter delta) usually describes a small distance, and the limit as delta goes to zero ($\delta \rightarrow 0$) describes the behaviour of the expression $h(\delta)$ for an infinitely short distance δ .

The SymPy function `limit` allows us to compute limit expressions. For example, if we want to see if the exponential function e^x or the polynomial function x^{100} grows faster in the limit as x goes to infinity, The code for computing the limit of the ratio between these two expressions is

```
>>> from sympy import limit, exp, oo
>>> limit(exp(x)/x**100, x, oo)
oo
```

code
2.4.24

The answer ∞ , written as `oo` (two lowercase letters “o”), tells us exponential functions grow faster than polynomial functions.

Limits are important in calculus because they are used in the formal definitions of the derivative and integral operations. The derivative is defined as a rise-over-run calculation for an infinitely short run. The integral is defined as a Riemann sum with infinitely narrow rectangles. We’ll explain both of these in the next sections.

Derivatives

The *derivative* function, denoted $f'(x)$, $\frac{d}{dx}f(x)$, or $\frac{df}{dx}$, describes the *rate of change* of the function $f(x)$. For example, the constant function $f(x) = c$ has derivative $f'(x) = 0$ since the function $f(x)$ does not change at all. The derivative function describes the *slope* of the graph of the function $f(x)$. The derivative of a line $f(x) = mx + b$ is $f'(x) = m$, since the slope of this line is equal to m . In general, the slope of a function is different at different values of x , so mathematicians invented a new notation for describing “the slope (rate of change) of the function $f(x)$ ” and obtained formulas for finding the derivative of any function.

The derivative function $f'(x)$ is defined as the rate of change of the function f at x :

$$f'(x) = \lim_{\delta \rightarrow 0} \frac{f(x + \delta) - f(x)}{\delta}.$$

In words, this formula describes the general rise-over-run calculation for computing the slope of a line that connects the points $(x, f(x))$ and $(x + \delta, f(x + \delta))$, with the step-length δ becoming infinitely small.

Geometrically, the derivative function computes the slope of the graph of the function $f(x)$ for all values of x . In general, the slope of a function is different for different values of x . Figure 2.52 shows the slope calculation for the function $f(x) = \frac{1}{2}x^2$ for two different values of x : $x = -0.5$ and $x = 1$.

The code below shows how to compute the derivative of the function $f(x) = mx + b$.

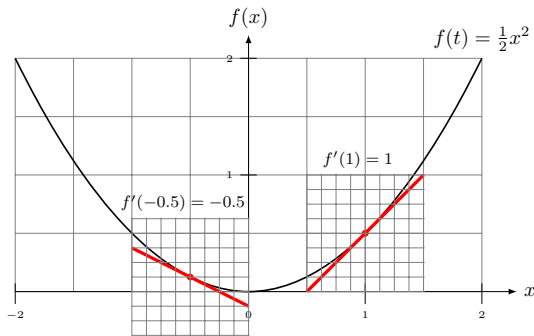


Figure 2.52: The derivative of the function at $x = a$ is denoted $f'(a)$ and describes the slope function at that point.

```
>>> from sympy import diff
>>> f = b + m*x
>>> diff(f, x)
m
```

In words, this calculation tells us the derivative of the function $f(x) = mx + b$ is the constant function $f'(x) = m$. The expression `diff(f, x)` tells SymPy to compute the derivative of the expression `f` with respect to the variable `x`.

Let's now define the function $f(x) = \frac{c}{2}x^2$ and compute its derivative.

```
>>> f = c/2 * x**2
>>> diff(f, x)
c*x
```

code
2.4.26

The derivative function is $f'(x) = cx$. See the plot in Figure 2.52 for an illustration of the case when $c = 1$.

Here is another example of a complicated-looking function f , that includes an exponential, a trigonometric, and a logarithmic function:

```
>>> from sympy import log, exp, sin
>>> f = exp(x) + sin(x) + log(x)
>>> diff(f, x)
exp(x) + cos(x) + 1/x
```

code
2.4.27

As you can see, using the function SymPy function `diff` allows you to compute the derivative function for any function $f(x)$.

Optimization algorithms

One of the most prominent applications of derivatives is *optimization*: the process of finding a function's maximum and minimum values.

Consider the shape of the function near a minimum value. The function is decreasing just before it reaches its minimum, and the

function increases just after its minimum. This means we can start at any point x_0 near the minimum and take “downhill” steps following the descending direction of the function, we’ll end up at the minimum value. This simple procedure that repeatedly takes steps in the direction where the function is decreasing turns out to be a very powerful tool that can find the minimum of any function. This procedure is called the *gradient descent algorithm*, where the name *gradient* refers to the derivative operation for multivariable functions.

In this book, we won’t discuss the details behind optimization algorithms, and instead rely on the computational tools available in *numpy*, *scipy*, and *sympy* to do optimization-type calculations for us. We’ll encounter optimization ideas (maximization and minimization) in several concepts in statistics: *maximum likelihood* and *least squares*, and rely on “visual proofs” for these optimization procedures. If you’re interested in attaining a deeper understanding of optimization algorithms, you can follow the links provided at the end of this section, but note such “under the hood” understanding is not required to continue with the rest of the book.

Here is a quick code example that shows how to use the function `minimize` defined in the module `scipy.optimize` to find the minimum value of the function $f(x) = (x - 5)^2$.

```
>>> from scipy.optimize import minimize
>>> def f(x):
    return (x-5)**2
>>> res = minimize(f, x0=0)
>>> res["x"][0] # = argmin f(x)
4.99999997455944
```

code
2.4.28

The `minimize` function takes two arguments: the function to minimize, and a initial value x_0 where to start the minimization process.

Riemann sums

The formal definition of the integration operation we learn in the first calculus course is based on the concept called a *Riemann sum*, which consists of approximating the area under the graph of a function using a series of rectangles.

The definite integral between $x = a$ and $x = b$ is defined as the limit of the following summation as n goes to infinity:

$$A(a, b) = \int_a^b f(x) dx \stackrel{\text{def}}{=} \lim_{n \rightarrow \infty} \sum_{k=1}^n f(a + k\Delta x) \Delta x.$$

This formula describes an approximation of the area under the graph of $f(x)$ with infinitely many rectangles. The height of each rectangle is given by $f(a + k\Delta x)$ and its width is $\Delta x = \frac{b-a}{n}$.

For example, let's take a look at how we can compute the integral of $f(x) = x^3 - 5x^2 + x + 10$ between $x = -1$ and $x = 4$. Figure 2.53 shows the graph of $f(x)$ (in red) and an approximation of the area under the graph of $f(x)$ as the sum of the areas of $n = 12$ rectangles.

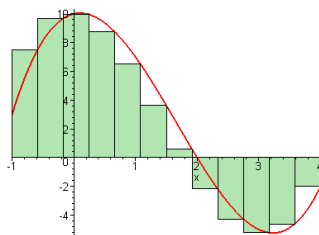


Figure 2.53: An approximation of the area under the function $f(x) = x^3 - 5x^2 + x + 10$ between $x = -1$ and $x = 4$ using $n = 12$ rectangles.

Note the approximation we obtain using $n = 12$ rectangles is not very precise: there are many rectangles sitting outside the area (overestimates), and other rectangles that don't cover the whole area (underestimates). But observe what happens when we use $n = 25$ and $n = 50$ rectangles, as shown in Figure 2.54. Now imagine how good the approximation will become when we use thousands or millions of individual rectangles. The Riemann sum formula computes an approximation to the area using an infinite number of rectangles.

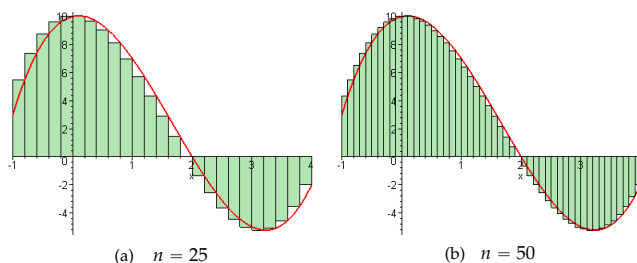


Figure 2.54: An approximation to the area under the graph of the function $f(x) = x^3 - 5x^2 + x + 10$ using $n = 25$ and $n = 50$ rectangles.

The Riemann sum is important as a *theoretical construct*, but nobody actually computes integrals by hand using this formula. The functions `quad` and `trapz` in the SciPy module `scipy.integrate` offer better tools for computing integrals, so you'll never have to use the Riemann sum formula above.

Fundamental theorem of calculus

The fundamental theorem of calculus (FTC) is a deep insight about the inverse relation that exists between the operations of integration $\int \cdot dx$ and differentiation $\frac{d}{dx}[\cdot]$.

The integral function $F_a(x)$ is obtained from the original function $f(x)$ using integration, $F_a(x) = \int_a^x f(u) du$. Another way to describe this is to say we *applied* the integration operator $\int \cdot dx$ on the function $f(x)$ to obtain the integral function $F_a(x)$. The derivative function $f'(x)$ is defined by the formula $f'(x) = \lim_{\delta \rightarrow 0} \frac{f(x+\delta) - f(x)}{\delta}$. We can also say we *applied* the derivative operator $\frac{d}{dx}[\cdot]$ to the function $f(x)$ to obtain the derivative function $f'(x)$. I use the word “operator” here to refer to an operation that acts on functions.

There is no reason *a priori* to think that integration and differentiation might be related: the former is a calculation about areas, while the latter is a calculation about slopes. The fundamental theorem of calculus reveals that they are in fact inverse operations: we can obtain the original function $f(x)$ from the integral function $F_a(x)$ by computing it’s derivative:

$$\frac{d}{dx}[F_a(x)] = \frac{d}{dx} \left[\int_a^x f(u) du \right] = f(x).$$

Note we used a temporary variable u as the integration variable, since x is already used to denote the upper limit of integration.

In order to understand the inverse relationship between integration and differentiation, we can draw an analogy with the inverse relationship between a function f and its inverse function f^{-1} , which *undoes* the effects of f . See Figure 2.44 on page 109. Given some initial value x , if we apply the function f to obtain the number $f(x)$, and apply the inverse function f^{-1} on the number $f(x)$, then the result will be the initial value x we started from:

$$f^{-1}(f(x)) = x.$$

Similarly, the derivative operator is the “inverse operator” of the integral operator. If you perform the integral operation followed by the derivative operation on some function, you’ll obtain the same function:

$$\text{diff}(\text{integrate}(f(x))) = f(x),$$

where we’ve used the SymPy functions `integrate` for computing the integrals, and `diff` (short for “differentiate”) for computing derivatives.

The code example below shows how we can construct a complicated-looking function `f` and compute its integral function `F` using SymPy.

```
>>> from sympy import diff, integrate, log, exp, sin
>>> f = log(x) + exp(x) + sin(x)
>>> F = integrate(f)
>>> F
x*log(x) - x + exp(x) - cos(x)
```

If we now take the derivative of the function F , we get back the original function f .

```
>>> diff(F)
log(x) + exp(x) + sin(x)
>>> diff(integrate(f)) == f # FTC part 1
True
```

code
2.4.30

The inverse relationship also holds for the opposite order of application: if we take the derivative of some function, then compute the integral of the derivative, then we arrive back at the original function (up to an additive constant factor).

```
>>> integrate(diff(f)) == f # FTC part 2
True
```

code
2.4.31

That's kind of cool, no?

In probability theory, the FTC tells us that the probability density can be obtained from the cumulative distribution using differentiation

$$f_X(x) = \frac{d}{dx}[F_X(x)] = \frac{dF_X}{dx}(x) = F'_X(x).$$

The fact that we can obtain f_X from F_X and vice versa, means we only need to define one of the two functions, and obtain the other function using differentiation or integration. In this book, we define the random variable X through its probability distribution function f_X , then define F_X as the integral of f_X . In other books, you might see the random variable X being defined through its cumulative distribution function F_X , with its probability density function f_X defined as the derivative of F_X .

Recommended calculus learning resources

There is an excellent free book called *Calculus made simple*[?] by Silvanus P. Thompson, which is a very friendly introduction to the subject. The subtitle of Thompson's book includes the phrase "Being a very-simplest introduction to those beautiful methods which are generally called by terrifying names," which should give you some idea about the author's attitude and the tone of his writing. You can also check out Chapter 5 in the *No bullshit guide to math and physics*[?], which is a compact introduction to calculus, and includes lots of examples from physics.

Above all, my advice is not to think of calculus as “advanced math theory” that might be difficult to understand, but instead as practical, useful math that allows you to do calculations—just look at the name of the thing! This means learning calculus is all about getting practical experience calculating limits, derivatives, and integrals of functions, which is best achieved by solving lots of problems. The problems and exercises in the books *Calculus made simple* and *No bullshit guide to math and physics* are therefore your best route for learning calculus, if you choose to pursue this subject.

Remember that you have SymPy at your disposal to solve calculus problems, so you don’t have to do all the calculations by hand using pen and paper. Indeed, you can solve any calculus problem using just a few lines of code using the SymPy functions `limit`, `diff`, and `integrate`. Check out Section III of the *SymPy tutorial*[?] to learn how to use these functions.

Exercises

TODO: 10x exercises on sets, functions, and integrals (geometric, numeric, and symbolic)

E2.34 Compute the integral function $F_0(b)$ for the function $f(x) = c$.

E2.35 Compute the integral function $G_0(b)$ for the function $g(x) = mx$.

TODO: 3 more

Links

[*Essence of calculus* series by 3Blue1Brown]

<https://www.youtube.com/playlist?list=PLZHQObOWTQDMsr9K-rj53DwVRMY03t5Yr>

TODO add more links to intro-calculus material

2.5 Continuous random variables

Continuous random variables represent smoothly varying quantities that are described by real numbers. The sample space \mathcal{X} for a continuous random variable consists of some subset of the real numbers. The probabilities of different outcomes are described by the *probability density function* f_X of the random variable X .

Examples of continuous random variables include the height and weight of individuals, physical measurements like time, length, area, volume, distance, etc. For example, the total volume of liquid that goes into a bottle produced at a bottling plant.

All the concepts and formulas you'll see in this section will sound like a "rerun" of the concepts and formulas we learned about discrete random variables in Section 2.1. Indeed, continuous random variables calculations are just like discrete random variable calculations, but we replace all summations with integrals. Since you already know about discrete random variables from Section 2.1, and you just learned about integration in the previous section, you're optimally prepared to learn about continuous random variables.

Don't rely on the "I've seen this before" feeling too much, though. Make sure you read all the definitions and formulas and know how exactly they are different from the equivalent formulas and equations we learned about in Section 2.1. This kind of review and study by comparison to concepts you've already seen is an excellent way to review what you've learned so far in the chapter. Trust me on this one: it's always a good idea to revisit all math concepts that you "know already" once in a while. The forgetting curve is real!

2.5.1 Definitions

Let's start with the definitions of continuous random variables. The *continuous random variable* X describes some smoothly varying quantity. We denote random variables by uppercase letters like X , Y , and Z , and particular *outcomes* of these random variables using lowercase letters like x , y , and z . The *sample space* \mathcal{X} (calligraphic X) is the set of all possible outcomes of the random variable X . In this section, we'll focus on *continuous* sample spaces, which consist of subsets of the real numbers. Examples of continuous sample spaces include all the real numbers $\mathbb{R} = (-\infty, \infty)$, the non-negative real numbers $\mathbb{R}_+ = [0, \infty)$, and the interval between zero and one $[0, 1]$.

The *probability density function* (pdf) is a function of the form $f_X : \mathcal{X} \rightarrow \mathbb{R}$ that tells us which regions of the sample space contain the likely outcomes of the random variable X . The outcome that the random variable X falls between a and b is denoted as the set $\{a \leq$

$X \leq b\}$, which is a subset of the sample space \mathcal{X} . The probability of the outcome $\{a \leq X \leq b\}$ is calculated by integrating the probability density function between a and b :

$$\Pr(\{a \leq X \leq b\}) = \int_{x=a}^{x=b} f_X(x) dx.$$

Intuitively, the integral of f_X from $x = a$ until $x = b$ calculates the total amount of density that lies in that interval of x -values. The probability density f_X varies for different values of x , so if we want to compute the total probability of X falling between $x = a$ and $x = b$, we must sum up (integrate) the total amount of f_X between $x = a$ and $x = b$.

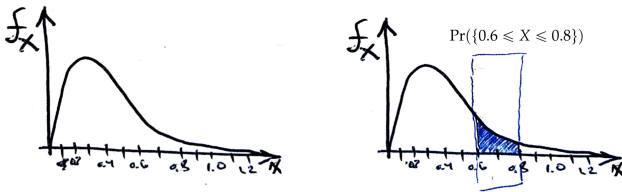


Figure 2.55: Example graph of the probability density function f_X of some random variable X . The area highlighted on the right shows the probability of the outcome $\{0.6 \leq X \leq 0.8\}$.

The left side of Figure 2.55 shows an example of a probability density function f_X for a continuous random variable X . The right side of the figure illustrates the calculation of the outcome that the variable X falls between 0.6 and 0.8, which we denote as the set $\{0.6 \leq X \leq 0.8\}$. We compute the probability of the outcome $\{0.6 \leq X \leq 0.8\}$ using the integral $\int_{x=0.6}^{x=0.8} f_X(x) dx$, which corresponds to the calculation of the area under the graph of the function f_X between $x = 0.6$ and $x = 0.8$.

Properties of probability density functions The probability density function f_X satisfies Kolmogorov's axioms of probability:

- Nonnegativity: $f_X(x) \geq 0$ for all $x \in \mathcal{X}$.
- Unit total: $\int_{x \in \mathcal{X}} f_X(x) dx = 1$.

The first axiom states that probability functions cannot take on negative values. The second axiom states that the total amount of probability over the whole sample space is 1. Note the similarity to the axioms for discrete distributions that we saw previously in Section 2.1. The only difference is that summation is replaced by integration.

Example 1: uniform distribution The *uniform distribution* assigns equal probability to all values in its sample space. The uniform distribution on the sample space $[0, 1]$ (the interval of real numbers between 0 and 1) is denoted $\mathcal{U}(0, 1)$ in math notation and `uniform(0, 1)` in code notation (more on that later).

The math shorthand statement “ $U \sim \mathcal{U}(0, 1)$ ” defines a random variable U that is distributed according to the uniform distribution $\mathcal{U}(0, 1)$. The symbol “ \sim ” is read “is distributed according to” in the context of probability theory. The random variable $U \sim \mathcal{U}(0, 1)$ is described by the following probability density function:

$$f_U(u) = \begin{cases} 1 & \text{if } 0 \leq u \leq 1, \\ 0 & \text{if } u < 0 \text{ or } u > 1. \end{cases}$$

The definition tells us that each outcome u between 0 and 1 is equally likely to occur, and values of u outside this range have zero probability of occurring. Figure 2.56 (a) shows the graph of the probability density function f_U . The total area under the graph of f_U is equal to 1, since the region is a square of width 1 and height 1.

The shaded region in Figure 2.56 (a) corresponds to the probability of the outcome of U will be between a and b , which we can denote as the set $\{a \leq U \leq b\}$. We know from geometry that the formula for the area of a rectangle of width w and height h is $w \cdot h$. Using this formula, we can calculate the probability of the outcome $\{a \leq U \leq b\}$ is $\Pr(\{a \leq U \leq b\}) = (b - a) \cdot 1$. For example, $\Pr(\{0.2 \leq U \leq 0.5\}) = (0.5 - 0.2) \cdot 1 = 0.3$, which means the interval $[0.2, 0.5]$ contains 30% of the total probability.

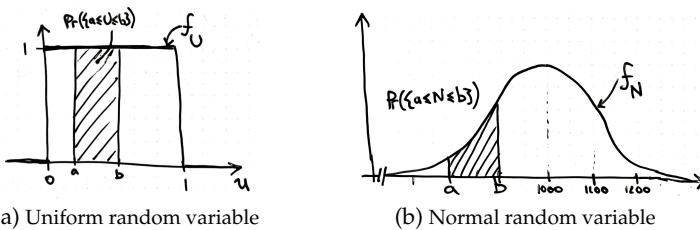


Figure 2.56: Probability density functions for two continuous random variables. The graph in (a) shows the probability density function f_U for the uniform random variable U with distribution $\mathcal{U}(0, 1)$. The graph in (b) shows the probability density function f_N for the normal random variable N with distribution $\mathcal{N}(1000, 100)$.

Example 2: normal distribution The family of normally distributed random variables is described by the notation $\mathcal{N}(\mu, \sigma)$,

where the constants μ (the Greek letter mu) and σ (the Greek letter sigma) are called *parameters* of the distribution. Normal variables are also called *Gaussian*. The random variable N with distribution $\mathcal{N}(\mu, \sigma)$ is described by the probability density function

$$f_N(n) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(n-\mu)^2}{2\sigma^2}}.$$

This complicated-looking formula describes a multi-step calculation in which we calculate the difference between n and the parameter μ , square the result, then divide by the constant $2\sigma^2$, pass the result through the function e^{-x} , and finally divide by the normalizing constant $\sigma\sqrt{2\pi}$. The combined result of all these transformations results in a function f_N that has a central peak at $n = \mu$ and rapidly decreases for values that “deviate” from this centre. The parameter σ controls the dispersion of the distribution.

Let’s consider a particular random variable N distributed according to the normal distribution with parameters $\mu = 1000$ and $\sigma = 100$. Figure 2.56 (b) shows the graph of the probability density function f_N of the random variable $N \sim \mathcal{N}(1000, 100)$. Recall “ \sim ” is math shorthand for the phrase “distributed according to.”

Calculating the area under the graph of f_N between a and b is not as straightforward as in the previous example, since the region doesn’t have a simple geometrical shape. To calculate the probability $\Pr(\{a \leq N \leq b\})$, we need to use the numerical integration techniques that we learned in the previous section. For example, using the function `scipy.integrate.quad`, we find $\Pr(\{800 \leq N \leq 900\}) = \int_{n=800}^{n=900} f_N(n)dn = 0.1359$, which corresponds to the area of the shaded region in Figure 2.56 (b). In words, this tells us the probability of observing N between 800 and 900 is 13.59%.

2.5.2 Cumulative distribution function

The *cumulative distribution function* of the random variable X describes the probability of outcomes that are smaller than or equal to some value b . The cumulative distribution function (CDF) is defined as the following integral:

$$F_X(b) \stackrel{\text{def}}{=} \Pr(\{X \leq b\}) = \int_{-\infty}^b f_X(x) dx.$$

Properties of the cumulative distribution function:

- If $b_1 \leq b_2$ then $F_X(b_1) \leq F_X(b_2)$, which means the function F_X is *nondecreasing*.

- $0 \leq F_X(b) \leq 1$ for all b .
- $\Pr(\{X > b\}) = 1 - F(b)$ for all b .

The first property follows from the *nonnegativity* property of the underlying probability density function $f_X(x) \geq 0$ for all $x \in \mathcal{X}$. The second property follows from the *unit total* property of the probability density function $\int_{-\infty}^{\infty} f_X(x)dx = 1$. The third property is a consequence of the general probability rule $\Pr(A^c) = 1 - \Pr(A)$, which applies for any outcome A , and since the outcome $\{X > b\}$ is the complement of the outcome $\{X \leq b\}$.

Knowing the cumulative distribution function F_X for the random variable X allows us to compute probabilities very quickly. We can find the probability of outcomes between a and b by computing the difference in the value of the cumulative distribution function:

$$\Pr(\{a \leq X \leq b\}) = \int_{-\infty}^b f_X(x)dx - \int_{-\infty}^a f_X(x)dx = F_X(b) - F_X(a).$$

Recall the visualization we showed in Figure 2.48 to better understand the above equation. The value $F_X(b)$ is the pre-computed integral until $x = b$, so if we want to know the value of the integral from $x = a$ to $x = b$, we can subtract the integral until $F_X(a)$.

Uniform random variable The cumulative distribution function for the uniform random variable $U \sim \mathcal{U}(0,1)$ is obtained through the following integral:

$$F_U(b) = \int_{u=0}^{u=b} f_U(u) du = \int_{u=0}^{u=b} 1 du = b, \quad \text{for all } b \in [0,1].$$

We can start the integration at $u = 0$ instead of $u = -\infty$ since f_U is zero for all $u < 0$. This integral corresponds to calculating the area under the graph of a constant function, which we saw previously in Figure 2.49. The region under the graph of f_U has the shape of a rectangle with height 1 and width b , so its area is $1 \cdot b = b$.

Figure 2.57 shows how $F_U(b)$ increases at a constant rate as b increases, until it reaches the maximum value when $b = 1$, and then stays constant after that.

Normal random variable The cumulative distribution function for the normal random variable $N \sim \mathcal{N}(1000,100)$ requires calculating the following integral:

$$F_N(b) = \int_{n=-\infty}^{n=b} f_N(n) dn = \int_{n=-\infty}^{n=b} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(n-\mu)^2}{2\sigma^2}} dn.$$

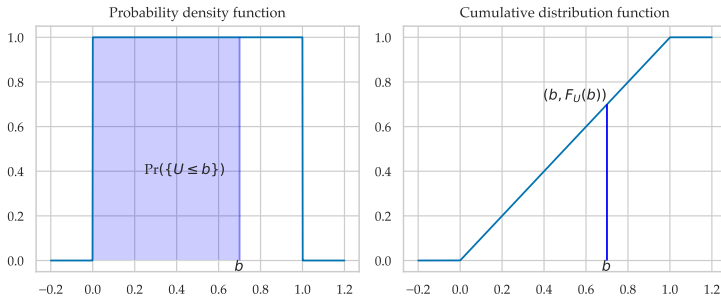


Figure 2.57: The left side shows the probability density function f_U , with the area from $x = 0$ until $x = b$ highlighted in blue. The right side shows CDF F_U with point at $(b, F_U(b))$ on the curve highlighted in blue.

There is no simple formula expression for the answer to this integral, so we need to use numerical integration (`scipy.integrate.quad`). See the graph of F_N in Figure 2.58, to get an idea of what it looks like.

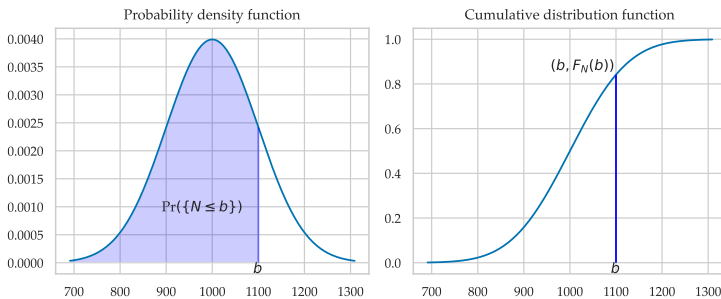


Figure 2.58: The left side shows the probability density function f_N with area from $x = -\infty$ until $x = b$ highlighted in blue. The right side shows CDF F_N with point at $(b, F_U(b))$ on the curve highlighted in blue.

Inverse of the cumulative distribution function

The inverse function of the cumulative distribution function is denoted $F_X^{-1}(q)$, and it helps us do “inverse probability” calculations. The inverse-CDF is also sometimes called the *percentile point function* or *quantile function*, since it tells us the positions of the quantiles of the random variable X . We specify the quantile $q \in [0, 1]$, and we want to find the value x_q such that the outcome $\{X \leq x_q\}$ will contain a proportion q of the total probability. In other words, x_q is the value such that $F_X(x_q) = q$.

For example, the 20% percentile (quantile $q = 0.2$) is the value of $x_{0.2}$ when the cumulative distribution reaches the value 0.2,

$F_X(x_{0.2}) = 0.2$. In other words, the interval $(-\infty, x_{0.2}]$ contains 20% of the total probability.

We can use the function F^{-1} to obtain a *confidence interval* that contain a certain percentage of the total probability. For example, the interval between $F_X^{-1}(0.025)$ and $F_X^{-1}(0.975)$ contains 95% of the probability density of the random variable X .

2.5.3 Calculating expectations

Suppose we're interested in calculating some value $w(X)$ that depends on the random variable X . You can think of w as a function of the form $w: \mathcal{X} \rightarrow \mathbb{R}$ that assigns different “winning” amounts to the different outcomes of the random variable X . Since the input to the function w is a random variable, the output value $w(X)$ is also a random variable. The *expected value* of the quantity $w(X)$ is obtained by computing the average value of $w(x)$ over all the possible outcomes x of the random variable X .

For a continuous random variable X , the expected value of $w(X)$ is defined as the following integral calculation:

$$\mathbb{E}_X[w(X)] \stackrel{\text{def}}{=} \int_{x \in \mathcal{X}} w(x) f_X(x) dx.$$

The expected value is computed by “weighing” each value of $w(x)$ by the probability $f_X(x)$ of the event $\{X = x\}$, “summed” over all possible outcomes for the random variable X .

Expectations play an important role in many calculations involving random variables. The mean of the random variable X is defined as the expectation $\mu_X \stackrel{\text{def}}{=} \mathbb{E}_X[X]$. The variance σ_X^2 is the expectation of $(X - \mu_X)^2$, and it is defined by the formula $\sigma_X^2 \stackrel{\text{def}}{=} \mathbb{E}_X[(X - \mu_X)^2]$. Let's talk about each of these in some more detail.

Measures of centre and dispersion

The *mean* of the continuous random variable X with probability density function f_X is given by

$$\mu_X \stackrel{\text{def}}{=} \mathbb{E}_X[X] = \int_{x \in \mathcal{X}} x \cdot f_X(x) dx.$$

The mean tells us the position of the distribution's centre of mass.

The *variance* of a continuous random variable is defined as

$$\sigma_X^2 \stackrel{\text{def}}{=} \mathbb{E}_X[(X - \mu_X)^2] = \int_{x \in \mathcal{X}} (X - \mu)^2 \cdot f_X(x) dx.$$

The variance calculates the average squared deviation of the random variable X from its mean μ_X , which is a measure of the dispersion of the distribution. The *standard deviation* of the random variable X is the square root of its variance $\sigma_X = \sqrt{\sigma_X^2}$.

Let's see some examples of mean and variance calculations.

Example 1 (cont.): mean and variance of the uniform distribution

The probability density function for the uniform random variable $U \sim \mathcal{U}(0, 1)$ is given by

$$f_U(u) = \begin{cases} 1 & \text{if } 0 \leq u \leq 1, \\ 0 & \text{if } u < 0 \text{ or } u > 1. \end{cases}$$

To calculate the mean of the random variable U , we need to compute the following integral:

$$\begin{aligned} \mu_U = \mathbb{E}_U[U] &= \int_{-\infty}^{\infty} u \cdot f_U(u) \, du \\ &= \int_0^1 u \cdot f_U(u) \, du \\ &= \int_0^1 u \cdot 1 \, du = \frac{1}{2}. \end{aligned}$$

This integral corresponds to the area of a triangle, see Figure 2.50 on page 115. In this case the upper limit is $b = 1$, so the formula becomes $\frac{1}{2}b^2 = \frac{1}{2} = 0.5$.

The formula for the variance is

$$\sigma_U^2 = \mathbb{E}_U \left[(U - \mu_U)^2 \right] = \int_0^1 \left(u - \frac{1}{2} \right)^2 \cdot f_U(u) \, du = \int_0^1 \left(u - \frac{1}{2} \right)^2 \cdot 1 \, du.$$

Let's use SymPy to calculate this integral

```
>>> from sympy import symbols, integrate
>>> u = symbols('u')
>>> integrate((u-1/2)**2 * 1, (u,0,1))
0.0833333333333333
```

code
2.5.1

The variance of U is equal to $0.08\bar{3} = \frac{1}{12}$. We can obtain the standard deviation of U by taking the square root of the variance.

```
>>> import numpy as np
>>> np.sqrt(0.0833333333333333)
0.2886751345948128
```

code
2.5.2

The random variable U therefore has mean $\mu_U = 0.5$ and standard deviation $\sigma_U = 0.289$.

Example 2 (cont.): mean and variance of a normal distribution

The probability density function for the normal random variable $N \sim \mathcal{N}(1000, 100)$ is given by

$$f_N(n) = \frac{1}{100\sqrt{2\pi}} e^{-\frac{(n-1000)^2}{2 \cdot 100^2}} = \frac{1}{100\sqrt{2\pi}} e^{-\frac{1}{2} \cdot \left(\frac{n-1000}{100}\right)^2}.$$

Let's create a Python function `fN` that corresponds to the math function f_N . We'll use the math functions `exp` and `sqrt` from the NumPy module to build up the complicated formula from simpler expressions.

```
>>> import numpy as np
>>> mu = 1000
>>> sigma = 100
>>> def fN(n):
    z = (n - mu)/sigma
    C = sigma * np.sqrt(2*np.pi)
    return 1 / C * np.exp(-1/2 * z**2)
```

code
2.5.3

Don't worry about the complicated-looking math and code. You won't have to write code like this every time you need to do some probability calculations. At the end of this section, we'll learn about the module `scipy.stats`, which will make working with probability distributions much easier. I'm just showing you the code for `fN` as an example of converting complicated-looking math equation into code.

To find the mean μ_N , we need to compute the expected value of N , which is defined as the integral

$$\mu_N = \mathbb{E}_N[N] = \int_{-\infty}^{\infty} n \cdot f_N(n) \, dn.$$

This integral doesn't correspond to a simple geometrical region, so we'll use numerical integration to calculate it.

```
>>> from scipy.integrate import quad
>>> def n_times_fN(n):
    return n * fN(n)
>>> muN = quad(n_times_fN, 0, 3000)[0]
>>> muN
1000.0
```

code
2.5.4

The function `n_times_fN` computes the value $n \cdot f_N(n)$, which we then integrate using the `quad` function between $a = 0$ and $b = 3000$. Note the mathematical equation describes an integration from minus infinity to plus infinity, but computing the integral over the finite interval $[0, 3000]$ provides an accurate approximation, since the value of f_N is negligible outside that interval.

The variance σ_N^2 is defined as the follows:

$$\sigma_N^2 = \mathbb{E}_N[(N - \mu_N)^2] = \int_{-\infty}^{\infty} (n - 1000)^2 \cdot f_N(n) \, dn.$$

We'll use a similar approach as in the code example above, based on the function `n_minus_mu_sq_times_fN` that computes the quantity $(n - \mu_N)^2 \cdot f_N(n)$. We'll then compute the integral to find the variance, and take the square root to find the standard deviation.

```
>>> def n_minus_mu_sq_times_fN(n):
    return (n-muN)**2 * fN(n)
>>> sigma_sq = quad(n_minus_mu_sq_times_fN, 0, 2000)[0]
>>> sigmaN = np.sqrt(sigma_sq)
>>> sigmaN
100.0
```

code
2.5.5

The random variable N has mean $\mu_N = 1000$ and standard deviation $\sigma_N = 100$.

A physical analogy Readers familiar with physics might recognize the formulas for the mean and the variance of a random variable are the same as the *centre of mass* and *moment of inertia* formulas in mechanics. For an object in one dimension, like a stick, the centre of mass x_{cm} represents the centre of its weight distribution. If you place your finger at x_{cm} , you can balance the stick using a single point of contact. See Figure ?? (page ??) for an illustration.

The *moment of inertia* of an object tells you how difficult it is to make the object turn around its centre of mass. Intuitively, the more the weight of an object is spread out away from its centre of mass, the more difficult it will be to make it turn. The contribution to the moment of inertia of a piece of mass is proportional to the *square* of its distance from the centre of mass. An object with small moment of inertia has most of its mass concentrated near x_{cm} . Similarly, the variance computes the average squared deviation from the mean. A distribution with small variance has most of its probability density concentrated close to the mean μ .

Measures of skewness and kurtosis

There are two additional properties that are useful for describing the “shape” of probability distributions: *skewness* and *kurtosis*, which are computed from “higher moments” of the distribution.

Moments of a distribution The formulas for the skewness and kurtosis are defined in terms of the *moments* of the distribution,

which are expectations of different powers of $(X - \mu_X)$. The m^{th} moment of the distribution f_X around its mean μ_X is defined as:

$$M_m \stackrel{\text{def}}{=} \mathbb{E}_X [(X - \mu_X)^m] = \int_{x \in \mathcal{X}} (x - \mu_X)^m \cdot f_X(x).$$

You're already familiar with this formula for the second moment ($m = 2$), which computes the variance of the distribution:

$$M_2 = \mathbb{E}_X [(X - \mu_X)^2] = \sigma_X^2.$$

Skewness The *skew* of the random variable X describes the lopsidedness of its distribution f_X —whether its peak lies to the left or to the right of its mean. We calculate the skewness $\text{skew}(X)$ in terms of a normalized third moment:

$$\text{skew}(X) = \frac{M_3}{\sigma_X^3}.$$

Figure 2.59 shows examples of distributions with positive, zero, and negative skew.

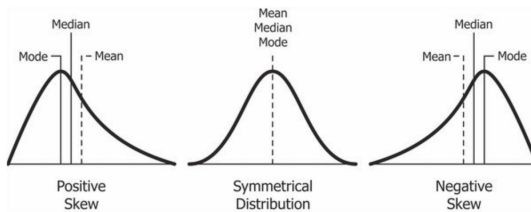


Figure 2.59: The distribution in the left panel has *positive skewness*, which means the distribution extends further to the right. We can also describe this distribution as *right skewed*. In the centre, we've shown a distribution with zero skewness, which is symmetric around the mean. Negative skewness, or *left skewness*, means the distribution extends further to the left.

Figure credit by Diva Jain CC BY-SA via Wikipedia commons File: Relationship_between_mean_and_median_under_different_skewness.png.

Recall we've already seen the concept of skewness informally, when we talked about data distributions. See Figure ?? on page ?? for examples of histograms that show positive skew (right skew), zero (symmetric), and negative skew (left skew).

Kurtosis The *kurtosis* is a measure of the “heaviness” of the tails of the distribution f_X . A distribution is *heavy-tailed* (sometimes called “fat-tailed”) if it has a lot of density in its tails, meaning far away

from its mean. The opposite is a *light-tailed* distribution with very thin tails that drop off to zero very quickly.

We calculate the *kurtosis* $\text{kurt}(X)$ in terms of a normalized fourth moment:

$$\text{kurt}(X) = \frac{M_4}{\sigma_X^4} - 3.$$

We'll explain where the constant -3 comes from in a few paragraphs. See Figure 2.78 on page 179 for an example of Student's *t*-distribution, which has "heavy tails," as compared to the distribution of the standard normal f_Z .

Example 2 (cont.): skewness and kurtosis of a normal distribution

Let's compute the skewness and the kurtosis of the *standard normal* random variable $Z \sim \mathcal{N}(\mu = 0, \sigma = 1)$. We'll start by creating a random variable rvZ from the normal model family imported from `scipy.stats`, initialized with parameters $\mu = 0$ and $\sigma = 1$. We can then call the method `rvX.stats()` to compute the mean, variance, skewness, and kurtosis in a single step:

```
>>> from scipy.stats import norm
>>> mu = 0          # position of the centre
>>> sigma = 1      # scale of dispersion
>>> rvZ = norm(mu, sigma)
>>> mean, var, skew, kurt = rvZ.stats(moments="mvsk")
>>> mean, var, skew, kurt
(      0,      1,      0,      0)
```

code
2.5.6

The keyword argument `moments="mvsk"` tells the `stats` method to compute all four moments of the distribution: *m* for the mean, *v* for the variance, *s* for the skewness, and *k* for the kurtosis. The first two numbers in the results show that the random variable Z has mean $\mu_Z = 0$ and variance $\sigma_Z^2 = 1^2 = 1$.

The third number tells us $\text{skew}(Z) = 0$, which makes sense since the normal distribution is symmetric.

The fourth number tells us the kurtosis is also zero $\text{kurt}(Z) = 0$. This is not a coincidence! Indeed, the formula for the kurtosis $\text{kurt}(X) = \frac{M_4}{\sigma_X^4} - 3$ is chosen specifically so that the normal distribution (the most common "shape" for data in the real world) will have zero kurtosis. In other words, the kurtosis $\text{kurt}(X)$ measures how "fat tailed" the distribution f_X is, *relative to* the tails of the normal distribution. Basically, the formula assumes the normal distribution is the reference for what is a "normal" amount of weight in the tails of a distribution. Distributions with positive kurtosis have tails heavier than the normal, and negative kurtosis indicates the tails of a distribution are lighter than the tails of the normal.

2.5.4 Computer models for random variables

The computer models defined in `scipy.stats` allow us to create random variable objects `rvX` distributed according to any probability distribution family. To create the random variable object `rvX`, we need to initialize the model by passing in a set of parameters: `rvX = <model>(<params>)`, where `<model>` is the name of the model family we imported from `scipy.stats`, and `<params>` is a comma-separated list of model-specific parameters.

Once we have created the random variable object `rvX`, we can use its methods (`.pdf(x)`, `.cdf(b)`, `.mean()`, `.std()`, etc.) to complete any probability calculations we might need. Table 2.2 lists all the methods available on any random variable `rvX` created from one of the families of pre-defined continuous probability distributions defined in `scipy.stats`.

method	args	math formula	description
<code>rvX.pdf</code>	<code>x</code>	$f_X(x)$	probability density function
<code>rvX.cdf</code>	<code>b</code>	$F_X(b)$	cumulative dist. function
<code>rvX.ppf</code>	<code>q</code>	$F_X^{-1}(q)$	inverse of the CDF function
<code>rvX.mean</code>		$\mu_X = \mathbb{E}_X[X]$	mean of the distribution
<code>rvX.var</code>		σ_X^2	variance of the distribution
<code>rvX.std</code>		σ_X	standard deviation
<code>rvX.median</code>		$F_X^{-1}(\frac{1}{2})$	median of the distribution
<code>rvX.support</code>		\mathcal{X}	bounds of the sample space
<code>rvX.interval</code>	<code>1-α</code>	$[F_X^{-1}(\frac{\alpha}{2}), F_X^{-1}(1-\frac{\alpha}{2})]$	$(1 - \alpha)$ confidence interval
<code>rvX.rvs</code>	<code>n</code>		generate n observations from X
<code>rvX.expect</code>	<code>w</code>	$\mathbb{E}_X[w(X)]$	expected value of $w(X)$

Table 2.2: Summary of the methods of continuous random variable objects `rvX` created from one of the model families in `scipy.stats`.

Compare Table 2.2 to the similar Table 2.1 (page 75) that shows the methods available on discrete random variable objects. Note most of the methods are common for both discrete and continuous random variables, except for the following two notable differences:

- Continuous random variables have a probability density function method `rvX.pdf(x)` instead of a probability mass function `rvX.pmf(x)`. This in turn dictates a different way to compute the probabilities of outcomes, using integration instead of summations. For example, to calculate the probability of the outcome $\{a \leq X \leq b\}$ for the continuous random variable `rvX`,

we can use the function `quad` from module `scipy.integrate`:
 $\Pr(\{a \leq X \leq b\}) = \text{quad}(\text{rvX.pdf}, a, b)$.

- The interpretation of the output of `rvX.support()` for a continuous random variable is a continuous interval—a subset of the real numbers. In contrast, the support of a discrete random variable is a range of whole numbers—a subset of the integers.

Let's now illustrate some of the “features” of computer models by working through a real-world example.

2.5.5 Kombucha volume example

Recall the Kombucha bottling example that we talked about in the introduction (see page 3). The math model for the volume of kombucha that goes into each bottle during the bottling process is $\mathcal{N}(1000, 100)$, which is a normally distributed random variable with mean $\mu = 1000$ and standard deviation $\sigma = 100$. We can build a computer model for the volume of kombucha by creating a random variable `rvN` based on the normal model `norm` initialized with parameters $\mu = 1000$ and $\sigma = 100$.

```
>>> from scipy.stats import norm
>>> rvN = norm(1000, 100)
```

code
2.5.7

The methods on the random variable object `rvN` will allow us to do all kinds of visualizations, probability calculations, and predictions.

Plotting the probability density function

Let's start by plotting the probability density function $f_N(n)$, which is available as the method `rvN.pdf(n)`.

```
>>> ns = np.linspace(0, 2000, 10000)
>>> fNs = rvN.pdf(ns)
>>> sns.lineplot(x=ns, y=fNs, label="pdf of $N$")
The result is shown in Figure 2.60.
```

code
2.5.8

To create the plot of f_N , we first create an array of input values for the interval $[0, 2000]$, then compute the values of the function f_N for these inputs, and finally call the Seaborn function `lineplot` to generate the plot.

Note we didn't need to write the complicated formula $\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(n-\mu)^2}{2\sigma^2}}$ for the probability density function $f_N(n)$ to obtain the plot in Figure 2.60. The method `rvN.pdf(n)` already contains the appropriate formula, with the parameters $\mu = 1000$ and $\sigma = 100$ that we specified when we created the object `rvN`.

The procedure for plotting the cumulative distribution function, $F_N(n) = \text{rvN.cdf}(n)$ is very similar.

code
2.5.9

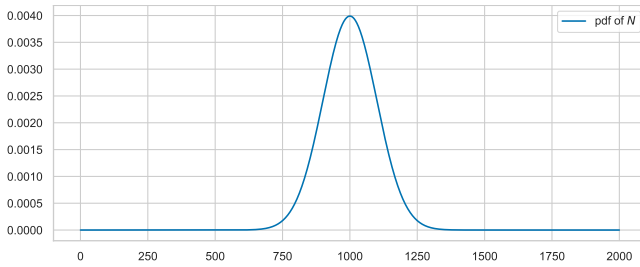


Figure 2.60: Probability density function f_N of the random variable rvN .

```
>>> ns = np.linspace(0, 2000, 10000)
>>> FNs = rvN.cdf(ns)
>>> sns.lineplot(x=ns, y=FNs, label="CDF of $N$")
The result is shown in Figure 2.61.
```

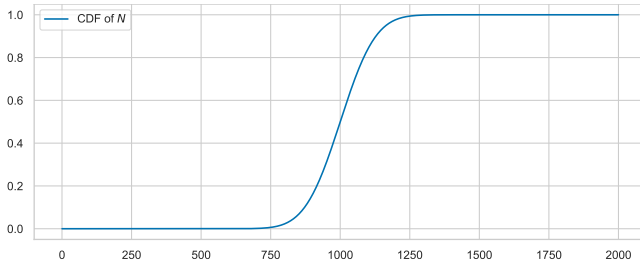


Figure 2.61: Cumulative distribution function F_N of the random variable N .

We didn't need to do any integration to obtain the plot in Figure 2.61. The method `rvN.cdf(b)` computed the integral $\int_{-\infty}^b f_N(n)dn$ for us.

Properties of the distribution

Let's verify that the properties of the computer model `rvN` are as expected: $\mu = 1000$ and $\sigma = 100$. These are the parameters we used when we created `rvN`, so we better get back the same values when we call the methods `rvN.mean()` and `rvN.std()`.

```
>>> rvN.mean()
1000.0
>>> rvN.std()
100.0
```

code
2.5.10

We didn't need to manually compute the integrals that correspond to $\mathbb{E}_N[N]$ and $\sqrt{\mathbb{E}_N[(N - \mu_N)^2]}$, because the methods `rvN.mean()` and `rvN.std()` did the calculations for us. Thank you Python. The median of the distribution is another property that is easy to obtain.

```
>>> rvN.median()
1000.0
```

The *support* of the distribution tells us where the sample space of the distribution starts and where it ends.

```
>>> rvN.support()
(-inf, inf)
```

code
2.5.12

In the case of normally distributed random variables, the sample space is all the entire real number line, from minus infinity to plus infinity.

* * *

Once we know the probability density function $f_N(n) = \text{rvN.pdf}(n)$ or the cumulative distribution function $F_N(b) = \text{rvN.cdf}(b)$, we can compute the probability of any outcome of interest for the random variable N . There are two principal questions we're interested in: probability calculations for certain outcomes like $\{a \leq N \leq b\}$, and inverse probability calculation to find the outcome that contains a desired proportion of the total probability (quantiles). We'll describe each of these tasks below.

Computing probabilities

Let's say someone asks you to compute the probability of the outcome $\{800 \leq N \leq 1200\}$, which is the probability that a bottle will contain between 800 ml and 1200 ml of kombucha. In math notation, we're interested in calculating $\Pr(\{800 \leq N \leq 1200\}) = \int_{n=800}^{n=1200} f_N(n)dn$, which is the integral of the probability density function $f_N = \text{rvN.pdf}$ between $n = 800$ and $n = 1200$. We can calculate this integral using the `quad` function from `scipy.integrate` as shown below:

```
>>> from scipy.integrate import quad
>>> quad(rvN.pdf, 800, 1200)[0]
0.954499736103641
```

code
2.5.13

The probability of the random variable N being between 800 and 1200 is 95.45%.

We can also obtain the same answer using the CDF function:

```
>>> rvN.cdf(1200) - rvN.cdf(800)
0.954499736103641
```

code
2.5.14

We don't need to remember the complicated-looking formula for the probability density function f_N or compute integrals, since the methods `rvN.pmf` and `rvN.cdf` take care of all the calculations for us.

Computing quantiles

Now let's discuss the inverse question—we want to find the interval $(-\infty, n_q]$ that contains the proportion q of the total probability. To answer this, we need to use the inverse-CDF function F_N^{-1} , which is provided by the point percentile method `rvN.ppf(q)`.

Recall the quartiles, percentiles, and quantiles calculations that we used to describe samples in Section ?? . The method `rvN.ppf(q)` is equivalent to computing the q^{th} quantile of the distribution. The `ppf` method allows us to compute the first quartile Q_1 as shown below.

```
>>> rvN.ppf(0.25)                                     code
932.55                                                  2.5.15
```

The interval $(-\infty, 932.55]$ contains 25% of the probability mass of the random variable N . We can verify this by doing the probability calculation $\Pr(\{N \leq 932.55\})$ using the `rvN.cdf()` method.

```
>>> rvN.cdf(932.5510249803918)                       code
0.25                                                    2.5.16
```

The second quartile Q_2 describes the median of the distribution—half the probability mass of the random variable lies to the left of Q_2 and the other half to the right of Q_2 .

```
>>> rvN.ppf(0.5)                                       code
1000.0                                                  2.5.17
```

The third quartile Q_3 tells us the value $n_{0.75}$ such that $\Pr(\{N \leq n_{0.75}\}) = 0.75$, and it is obtained by calling the `rvN.ppf` method with the input $q = 0.75$.

```
>>> rvN.ppf(0.75)                                       code
1067.45                                                  2.5.18
```

Computing the tails of the distribution

Another calculation we're often interested in is to find the *tails* of the distribution, which contain the most extreme or unexpected values. For example, the 5% *left tail* of the distribution is the interval $(-\infty, n_{q_l}]$, where n_{q_l} is the position of the $q_l = 0.05^{\text{th}}$ quantile of the distribution. We compute the cutoff value n_{q_l} of the 5% left tail of the distribution using:

```
>>> rvN.ppf(0.05)                                       code
835.514                                                  2.5.19
```

The above calculation tells us that the interval $(-\infty, 835.514]$ contains 5% of the probability. In other words, the outcome $\{N < 835.514\}$ has less than 5% chance of occurrence.

The 5% *right tail* of the distribution is the interval $[x_{q_r}, \infty)$, where n_{q_r} is the value of the $q_r = 0.95^{\text{th}}$ quantile of the distribution. We compute it as follows.

```
>>> rvN.ppf(0.95)
1164.485
```

code
2.5.20

The interval $(-\infty, 1164.485]$ contains 95% of the probability, so the remaining interval $[1164.485, \infty)$ contains only 5% of the probability. In other words, the probability of observing extremely high values $\{N > 1164.485\}$ is less than 5%.

Computing confidence intervals

The concept of a *confidence interval* (CI) combines the left-tail and right-tail calculations to define a centre-interval $[n_{q_l}, n_{q_r}]$ that contains the *bulk* of the distribution. A 90% confidence interval is defined as the interval $[n_{q_l}, n_{q_r}]$ that contains 0.9 of the total probability: $\Pr(\{n_{q_l} \leq N \leq n_{q_r}\}) = \int_{n_{q_l}}^{n_{q_r}} f_N(n)dn = 0.9$.

We can obtain the 90% confidence interval for the random variable `rvN` by combining the $q = 0.05$ and $q = 0.95$ quantiles of the distribution obtained from the method `rvN.ppf(q)`.

```
>>> rvN.ppf(0.05), rvN.ppf(0.95)
(835.514, 1164.485)
```

code
2.5.21

The 90% confidence interval we obtain is $[F_N^{-1}(0.05), F_N^{-1}(0.95)] = [835.514, 1164.485]$. This means, if we were to generate thousands or millions of observations from the random variable N , 90% of these observations will fall in that interval.

The method `rvN.interval()` is a shortcut for computing the confidence interval in a single step:

```
>>> rvN.interval(0.90)
(835.514, 1164.485)
```

code
2.5.22

The method `rvN.interval(p)` computes $[F_N^{-1}(\frac{1-p}{2}), F_N^{-1}(1 - \frac{1-p}{2})]$.

Confidence intervals are often defined in terms of the parameter $\alpha = 1 - p$, which describes the “missing” probability—the weight of the tails of the distribution that are not included in the confidence interval. The $(1 - \alpha)$ -confidence interval is defined as $[F_N^{-1}(\frac{\alpha}{2}), F_N^{-1}(1 - \frac{\alpha}{2})]$, and it is computed using `rvN.interval(1 - α)`.

Generating random observations

Let’s say you want to generate 10 observations from the random variable N . You can do this by calling the method `rvN.rvs(10)` as shown below.

code
2.5.23

```
>>> ns = rvN.rvs(10)
>>> ns
[1178.86, 1043.65, 1009.65, 813.65, 972.26, 964.52,
 991.73, 937.3, 995.62, 952.28]
```

To compute the mean of these 10 observations, we can use the Python function `sum(ns)`, which computes the sum of the values in the list `ns`, then divide the sum by the length of the list, which we can obtain by calling `len(ns)`.

```
>>> ns_mean = sum(ns) / len(ns)
>>> ns_mean
985.95
```

code
2.5.24

Note the mean of this sample of 10 observations from `rvN` is close to the mean of the distribution `rvN.mean()`, but it's not identical. This example touches on a very interesting topic, which is the variability of the statistics we observe when we study samples generated from a given distribution. The entire Section 2.8 is dedicated to this topic.

Computing expectations

Suppose the distributor that purchases the kombucha bottles has a policy requiring the volume of kombucha in each bottle to be within two standard deviations of the mean. Bottles that contain between 800 ml and 1200 ml will be accepted, and you'll receive a payment of \$2 for each bottle. Bottles outside that range get rejected, and you don't get paid for them. Given the model for the variability of the kombucha bottling process, can you compute the expected payment you'll receive per 100 bottles?

To answer this question, we'll first build a function that describes the payment per bottle, as a function of the volume. If the bottle fits the distributor's "spec" you get paid \$2, else the payment is zero.

```
>>> def payment(n):
    if 800 <= n and n <= 1200:
        return 2
    else:
        return 0
```

code
2.5.25

We can verify the function `payment` works as expected by testing an input that is "in spec" and another input that is "out of spec."

```
>>> payment(1050)
2
>>> payment(1250)
0
```

code
2.5.26

We can now compute the expected value of the payment function under the random variable N using the `rvN.expect()` method.

```
>>> rvN.expect(payment, lb=0, ub=2000)
1.9090
```

code
2.5.27

The code above calculates the expectation $\mathbb{E}_N[\text{payment}(N)]$, which is the value of the expected payment per bottle. Note we manually specified the lower bound $\text{lb}=0$ and upper bound $\text{ub}=2000$ for computing the expectation, which covers most of the weight of the distribution. We can expect to get paid \$190.90 for each batch of 100 bottles delivered to the distributor.

* * *

All the above code calculations used the random variable `rvN` based on the normal model `norm`, which is defined in the module `scipy.stats`. In Section 2.6, we'll learn about other computer models for continuous distributions that are defined in `scipy.stats`. Here is a sneak-peek preview of the other computer models we'll use later in the book: `uniform`, `gamma`, `expon`, `t`, `chi2`, and `f`. Each of these models provides the same set of methods for computing probabilities, quantiles, and confidence intervals as we saw in the above example. See Table 2.2 for the complete list of methods.

Exercises

E2.36 Compute the expected value $\mathbb{E}_U(w(U))$, where $w(u) = u^3$ and U is the uniform random variable $\mathcal{U}(0, 1)$.

2.5.6 Multiple continuous random variables

Let's now talk about two continuous random variables (X, Y) defined over the joint sample space $\mathbb{R} \times \mathbb{R}$. This section will be a “rerun” of what we learned about multiple random variables in Section 2.2, but this time X and Y are continuous random variables.

Definitions

Consider the pair of random variables (X, Y) defined in the *joint sample space* $\mathcal{X} \times \mathcal{Y}$. A particular *outcome* in the joint sample space looks like a pair of real numbers (x, y) . Geometrically speaking, the joint sample space is a two-dimensional region.

Joint probability density function

The *joint probability density function* f_{XY} is the main tool for modelling relationships between two random variables X and Y . By choosing the appropriate function f_{XY} , we can describe and model various relationships between the two random variables.

Since the probability density function $f_{XY}(x, y)$ has two inputs, we can plot its graph as a three-dimensional surface, as shown in Figure 2.62 (a). The height of the surface is given by $f_{XY}(x, y)$ for each coordinate (x, y) in the two-dimensional sample space.

We can also represent the probability density function f_{XY} using a *contour plot*, in which darker shaded regions indicating higher probability density, as shown in Figure 2.62 (b). The contour plot allows us to see more clearly the joint variability of the random variables X and Y : larger X values are associated with larger Y values, so it seems there is a linear relationship between these two variables. We'll learn how to quantify the strength of such relationship using the concepts of *covariance* and *correlation* later in this section.

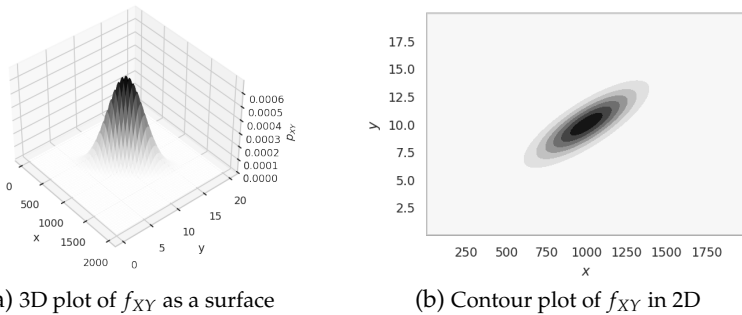


Figure 2.62: Graphical representations of the joint probability density function f_{XY} . In (a) we plot the graph of f_{XY} as a surface in a three-dimensional space. In (b) we see a plot of f_{XY} in two dimensions (as if looking from above), and represent the third dimension using shading. The darker shaded regions correspond to locations with higher probability density.

The exact formula for the probability density function $f_{XY}(x, y)$ shown in Figure 2.62 is not important. I've just chosen a multivariate distribution (multivariate normal), with an interesting density distribution for illustrative purposes.

The outcomes in the joint sample space consists of pairs of outcomes. For example, the outcome where X takes on a value between $x = a$ and $x = b$, and Y takes on a value between $y = c$ and $y = d$, is written as $\{a \leq X \leq b, c \leq Y \leq d\}$ in set notation. Another, more compact representation of this outcome is as a product of intervals $A = [a, b] \times [c, d]$. Geometrically speaking, this outcome describes a rectangular region with width $b - a$ and height $d - c$. The probability of the outcome A is obtained using the following double

integration calculation:

$$\Pr(A) = \int \int_{(x,y) \in A} f_{XY}(x,y) \, dx dy = \int_{x=a}^{x=b} \int_{y=c}^{y=d} f_{XY}(x,y) \, dx dy.$$

In words, this integral describes the process of adding up the values of the probability density function $f_{XY}(x,y)$ over the two-dimensional region A .

The concept of a joint probability density function f_{XY} is directly analogous to the single-variable probability density functions, which you're already familiar with. Yes there are more dimensions now, so instead of single-variable integration we need double-variable integration, but this is not a big deal. You have to trust me on this one—double integral formulas look intimidating, but there is nothing fancy going on. It's still the same idea of calculating the “total” amount of f_{XY} over some region of integration.

Speaking of totals, let's review the mathematical axioms that apply to all joint probability density functions f_{XY} :

- Nonnegativity: $f_{XY}(x,y) \geq 0$ for all x and y .
- Unit total: $\int \int_{(x,y) \in \mathcal{X} \times \mathcal{Y}} f_{XY}(x,y) \, dx dy = 1$.

The unit total property describes the calculation of the integral of f_{XY} over the entire sample space, which corresponds to the integral $\int_{x=-\infty}^{x=\infty} \int_{y=-\infty}^{y=\infty} f_{XY}(x,y) \, dx dy = 1$.

Example 3: product of uniform distributions Consider the uniformly distributed random variables $U \sim \mathcal{U}(0,100)$ and $V \sim \mathcal{U}(0,10)$. The joint probability density function that describes the pair (U,V) can be written as the product of the probability density functions of the two random variables:

$$f_{UV}(u,v) = f_U(u) \cdot f_V(v),$$

where $f_U(u) = \frac{1}{100}$ for $u \in [0,100]$ and zero otherwise, and $f_V(v) = \frac{1}{10}$ for $v \in [0,10]$ and zero otherwise.

Figure 2.63 shows two plots of the joint probability density function f_{UV} .

Marginal distribution functions

The *marginal probability density function* f_X is obtained from the joint probability density function f_{XY} by integrating over all possible values for the variable y :

$$f_X(x) = \int_{y=-\infty}^{y=\infty} f_{XY}(x,y) \, dy.$$

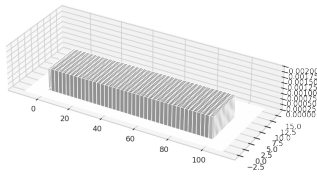
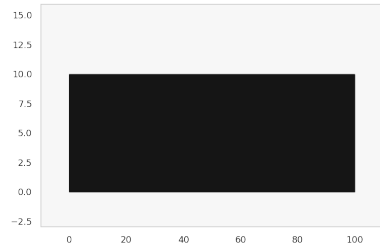
(a) 3D plot of f_{UV} as a surface(b) Contour plot of f_{UV} in 2D

Figure 2.63: Graphical representations of the joint probability density function f_{UV} . The graph of f_{UV} looks like a raised rectangle of width 100 along the u -axis, and width 10 along the v -axis.

The idea for a marginal distribution f_X is to get rid of the Y randomness and corresponds to a description of the random variable X when the random variable is unknown. The name *marginal distribution* comes from the procedure we use to compute it, by “summing” over all y values for a given x and writing the total in the margin. See Figure 2.20 for an illustration, and also recall Table ?? on page ??, where we used a similar procedure for computing the marginal frequencies in a two-way table.

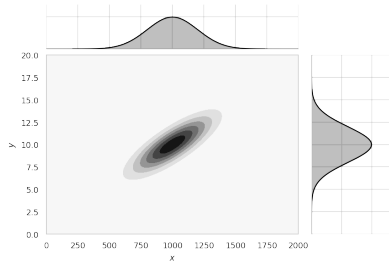


Figure 2.64: The top panel shows the marginal distribution f_X , which is obtained by integrating f_{XY} over all values of Y . The right panel shows the marginal distribution f_Y obtained by integrating f_{XY} over all values of X .

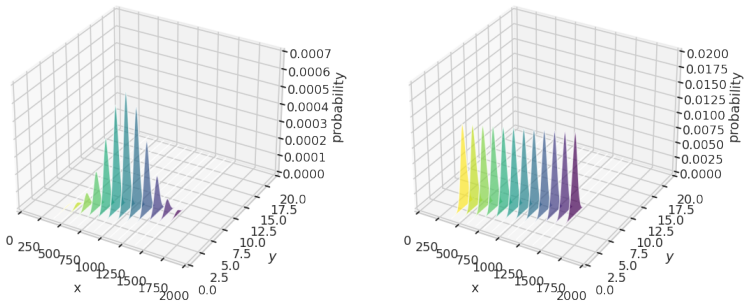
The marginal distributions f_Y is obtained from the joint distribution f_{XY} similarly, but this time integrating over all possible values for the variable x : $f_Y(y) = \int_{-\infty}^{\infty} f_{XY}(x, y) dx$. The marginal distribution f_Y describes the randomness of Y when we don’t know the value of X .

Conditional probability distributions

The *conditional probability density functions* $f_{X|Y}$ and $f_{Y|X}$ are defined as follows:

$$f_{X|Y}(x|y) = \frac{f_{XY}(x,y)}{f_Y(y)} \quad \text{and} \quad f_{Y|X}(y|x) = \frac{f_{XY}(x,y)}{f_X(x)}.$$

The vertical bar is pronounced “given” and describes situations where the realization of some random variables is known. For example, the conditional distribution $f_{Y|X}(y|x_a)$ describes the probabilities of the random variable Y , given we know the value of the random variable X is x_a . In general, there is a different conditional distribution $f_{Y|X}$ for each of the possible values of $x \in \mathcal{X}$.



(a) Slices through $f_{XY}(x, y)$ at different values of the variable x .

(b) Conditional distributions $f_{Y|X}(y|x)$ for different values of x

Figure 2.65: Conditional probability distributions $f_{Y|X}$ are obtained by slices of the joint probability distribution along different values of x . The left graph shows the shape of the slices before normalization. The right side shows the normalized slices, which are proper probability distributions.

Example 4: Kombucha volume increasing with temperature

One of your production line engineers in the Kombucha brewery, has a theory that the volume that goes into each bottle depends on the ambient temperature in the room. They have observed bottles get under-filled on cold days, and over-filled on hot days.

In order to model the temperature-dependence of the kombucha volume, we need to consider now the joint sample space (N, T) , where T describes the ambient temperature in the room, and N describes the volume that goes into each bottle.

Suppose the temperature random variable is normally distributed with standard deviation $\sigma_T = 2$ around the mean of $\mu_T =$

20, which is written mathematically as $T \sim \mathcal{N}(20, 2)$. Using the estimates provided by the production engineer for the increase in the average volume of kombucha as a function of the temperature t , we can build a model for the random variable $N \sim \mathcal{N}(\mu_N, 75)$, where $\mu_N = 1000 + 35(t - 20)$. In other words, the conditional distribution $f_{N|T}$ is $\mathcal{N}(1000 + 35(T - 20), 75)$.

By studying the dependence between the bottling temperature T and the volume of kombucha N , you hope you'll be able to improve the reliability of the kombucha bottling process. Recall that your distributor only pays for bottles that are within "spec" (between 800 ml and 1200 ml).

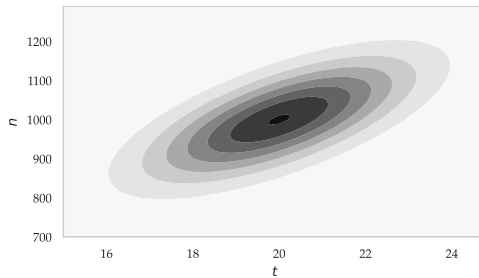


Figure 2.66: Graph of the joint probability density function f_{NT} . The kombucha volume random variable N tends to increase as the temperature increases.

Looking at shape of the joint probability density function in Figure 2.66, we see if we want the kombucha volume to fall within the interval $[800, 1200]$, we should keep the production facilities at $t = 20$.

Example 5: Temperature-dependent variance

Your master brewer has a different theory. According to her, the effects of temperature is to create more bubbles, and so the variability in the kombucha volume increases. In other words, it's the variance that depends on the temperature, not the mean.

Based on their parameter estimates provided by the master brewer, you build the model for the volume of kombucha $N \sim \mathcal{N}(1000, \sigma_N)$, where $\sigma_N = 100 + 5(t - 20)$. In other words, the conditional distribution $f_{N|T}$ is $\mathcal{N}(1000, 100 + 5(T - 20))$. We assume the temperature random variable is $T \sim \mathcal{N}(20, 2)$.

Figure 2.67 shows the variance of kombucha volume increases with temperature. Note we have plotted the temperature on the horizontal axis, to reinforce the fact we're looking at the dependence of N on T .

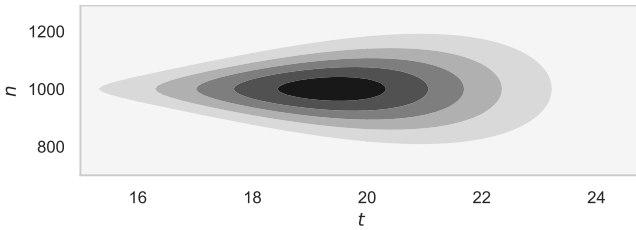


Figure 2.67: Graph of the joint probability density function f_{NT} . Note the variance of the kombucha volume random variable N tends to increase with temperature.

According to the contour plot in Figure 2.67, the colder the bottling temperature, the lower the variability, so you should aim to cool the kombucha as much as possible, if we want most of the volumes to fall in the distributor's requirements $N \in [800, 1200]$.

* * *

We just saw two different models for the dependence between temperature T and kombucha volume N . Using math equations (and computer calculations), we were able to come up with different theoretical models, and arrive at two possible strategies for improving the production process. Unfortunately, the two theoretical models suggest different strategies! According to the model suggested by the production engineer (Example 4), we should run the bottling process at 20 degrees for optimal results. The master brewer's model we developed in Example 5 suggests we should do the bottling at low temperature, like 16 degrees. What is the right thing to do? Should you believe the model from Example 4 or from Example 5?

These examples were chosen to illustrate the point that mathematical modelling cannot lead to answers on its own. Math models are only useful when we connect them to the real-world. If you want to know the optimal temperature, you'll need to check which of the two math models are better description of real-world data. This is what statistics is all about. In chapters ?? and ??, we'll learn statistical procedures based on real-world data observations that allow us to choose the "best fit" probability model for a given situation.

Who knows, maybe the production engineer and the master brewer are both "right" and the best model for describing the dependence between the random variables N and T combines aspects of temperature dependence on both the mean and the variance of N .

Useful probability formulas

Let's revisit the probability formulas we saw in Section 2.2 that are most helpful for doing calculations with multiple random variables.

Chain rule formula The chain rule states that we can decompose the joint distribution f_{XY} as the product of the conditional distribution $f_{Y|X}$ and marginal distribution f_X :

$$f_{XY}(x, y) = f_{Y|X}(y|x) \cdot f_X(x).$$

In words, this means the joint-uncertainty in (X, Y) can be broken down into X -uncertainty, and Y -given- X -uncertainty. This formula follows from the definition of conditional probability distribution $f_{Y|X}(y|x) \stackrel{\text{def}}{=} \frac{f_{XY}(x, y)}{f_X(x)}$. We can also write the joint probability density function $f_{XY}(x, y)$ as the product $f_{X|Y}(x|y) \cdot f_Y(y)$.

Bayes' rule formula Bayes' rule is a useful formula based on the chain rule:

$$f_{Y|X}(y|x) = \frac{f_{X|Y}(x|y)f_Y(y)}{\int_{y' \in \mathcal{Y}} f_{X|Y}(x|y')f_Y(y')dy'}.$$

Note we're using a different integration variable y' in the denominator to avoid confusion with the variable y used in the numerator.

I know the above formulas for Bayes rule for continuous random variables looks complicated and intimidating, but you look at it for a little while, you'll see it's quite simple. Recall that the primary use of Bayes' rule is to convert our knowledge of $f_{X|Y}$ into knowledge about $f_{Y|X}$. This simple-sounding task, turns out to be extremely powerful. Indeed, we could say that 50% of all statistics, and 70% of all machine learning ideas currently in use in research and in industry are consequences of this simple formula.

Multivariable expectation

Consider the function $w(X, Y)$ which depends on the random variables X and Y . The expected value of w is defined as

$$\mathbb{E}_{XY}[w(X, Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w(x, y) f_{XY}(x, y) dx dy.$$

This is exactly the same formula as the expectation for a single variable, but we now have two-dimensional sample space, so computing the expected value of $w(X, Y)$ requires a double integral.

Covariance The *covariance* of the random variables (X, Y) is defined as:

$$\begin{aligned}\mathbf{cov}(X, Y) &= \mathbb{E}_{XY}[(X - \mu_X)(Y - \mu_Y)] \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (x - \mu_X)(y - \mu_Y) f_{XY}(x, y),\end{aligned}$$

where $\mu_X = \mathbb{E}_X[X]$ and $\mu_Y = \mathbb{E}_Y[Y]$ are the means of the marginal distributions f_X and f_Y . In words, the covariance $\mathbf{cov}(X, Y)$ measures the joint variability of two random variables X and Y .

An alternative formula for calculating the covariance is:

$$\mathbf{cov}(X, Y) = \mathbb{E}_{XY}[XY] - \mu_X \mu_Y.$$

In words, this calculation tells us the covariance $\mathbf{cov}(X, Y)$ can also be computed as the expectation of the product XY minus the product of the means of the marginals.

Correlation The *correlation* between the random variables X and Y is denoted $\mathbf{corr}(X, Y)$ or ρ_{XY} . The correlation between X and Y is defined as the ratio of the covariance $\mathbf{cov}(X, Y)$ to the product of the variables' standard deviations:

$$\mathbf{corr}(X, Y) = \frac{\mathbf{cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Dividing the covariance by the product of the standard deviations $\sigma_X \sigma_Y$ has a normalizing effect, constraining the correlation $\mathbf{corr}(X, Y)$ to always be between -1 and 1 .

Recall we've already the concepts of covariance and correlation twice already, earlier in Section ?? (correlation between two numerical variables in a dataset) and again in Section 2.2 (correlation between two discrete random variables), so consider all the above formulas as reminders: we simply replaced summations with integrals, the but logic is the same.

All the formulas we learned about expectations and variance calculations for multivariate discrete variables also apply to multivariate continuous random variables. For example, $\mathbb{E}_{XY}[X + Y] = \mathbb{E}_X[X] + \mathbb{E}_Y[Y]$, and $\mathbf{var}(X + Y) = \mathbf{var}(X) + \mathbf{var}(Y) + 2\mathbf{cov}(X, Y)$. Jump back to page 53 to review the properties and formulas of the expectation operator \mathbb{E}_{XY} and the covariance $\mathbf{cov}(X, Y)$.

Example n: compute the correlation and covariance of f_{NT} TODO

Independent random variables Two random variables X and Y are called *independent* if their joint probability density function can be written as the product of the marginal distributions:

$$f_{XY}(x, y) = f_X(x)f_Y(y).$$

The randomness of X does not depend on the randomness of Y , and vice versa.

When X and Y are independent random variable, the conditional distribution function $f_{Y|X}$ is equal to the marginal distribution f_Y . In other words, knowing the value of X doesn't change anything about our uncertainty of Y . Similarly, the conditional distribution $f_{X|Y}$ is equal to the marginal f_X . The covariance is zero $\text{cov}(X, Y) = 0$, and by extension the correlation is also zero $\text{corr}(X, Y) = 0$.

Example 6: sum two normal random variables Consider the random variables $Z_1 \sim \mathcal{N}(0, 1)$ and $Z_2 \sim \mathcal{N}(0, 1)$, which are both instances of the standard normal Z . Define the random variable which is their sum $S = Z_1 + Z_2$. In words, S describes the sum of two independent standard normal random variables.

Using the fact that Z_1 and Z_2 are independent, we can easily compute the mean of the random variable S :

$$\mu_S \stackrel{\text{def}}{=} \mathbb{E}_S[S] = \mathbb{E}_{XY}[X + Y] = \mathbb{E}_X[X] + \mathbb{E}_Y[Y] = \mu_Z + \mu_Z = 0.$$

This is not too surprising, since the means of the Z_1 and Z_2 are zero, it makes sense that their sum is also zero.

Let's not calculate the variance of the random variable S .

$$\text{var}(S) = \text{var}(Z_1 + Z_2) = \text{var}(Z_1) + \text{var}(Z_2) = 2 \cdot \text{var}(Z) = 2.$$

The variance of S is double the variance of the individual random variables $\sigma_S^2 = 2\sigma_Z^2 = 2$.

Independent, identically distributed setting Consider the sequence of n independent observations from the random variable X , which we'll write as a sequence (X_1, X_2, \dots, X_n) . Each X_i is a copy of the same continuous probability random variable X , described by the probability density function f_X . This is the *independent, identically distributed* setting (*i.i.d.* for short).

The joint probability distribution for the sequence (X_1, X_2, \dots, X_n) is the product of n copies of the probability distribution of the random variable X :

$$f_{X_1 X_2 \dots X_n}(x_1, x_2, \dots, x_n) = f_X(x_1)f_X(x_2) \cdots f_X(x_n).$$

This *product structure* of the joint distribution $f_{X_1 X_2 \dots X_n}$ tells us the random variables are *independent*, and each X_i is an *identical* copy of the random variable $X \sim f_X$.

We'll study the properties of these sequences of n observations from the same random variable later in Section 2.8.

Example 7: average of n copies of the standard normal Consider the sequence n independent observations from the standard normal (Z_1, Z_2, \dots, Z_n) , where each $Z_i \sim \mathcal{N}(0, 1)$. Define the random variable A which computes the average value of the sequence (Z_1, Z_2, \dots, Z_n) :

$$A = \frac{1}{n} \sum_{i=1}^n Z_i = \frac{1}{n} (Z_1 + Z_2 + \dots + Z_n).$$

We can use the i.i.d. properties of A to obtain its mean μ_A and

$$\mathbb{E}_A[A] = \mathbb{E}_{Z_1 Z_2 \dots Z_n} \left[\frac{1}{n} (Z_1 + Z_2 + \dots + Z_n) \right] = \frac{1}{n} \cdot n \cdot \mathbb{E}_Z[Z] = \mu_Z.$$

Since the mean of the standard normal is zero $\mu_Z = 0$, the mean of the average of n i.i.d. copies of Z is also zero $\mu_A = 0$.

More interestingly, let's compute the variance of A :

$$\begin{aligned} \sigma_A^2 &= \mathbf{var}(A) = \mathbf{var} \left(\frac{1}{n} (Z_1 + Z_2 + \dots + Z_n) \right) \\ &= \frac{1}{n^2} \cdot \mathbf{var}(Z_1 + Z_2 + \dots + Z_n) \\ &= \frac{1}{n^2} \cdot n \cdot \mathbf{var}(Z) \\ &= \frac{1}{n} \cdot \mathbf{var}(Z). \end{aligned}$$

This calculation tells us the variance of the average of n random normally distributed variables is equal to $\frac{1}{n}$ times the variance of the individual random variables:

$$\sigma_A^2 = \frac{1}{n} \cdot \sigma_Z^2.$$

This may seem like a counterintuitive result. Remember that the random variable A contains the randomness from n standard normals, and one might think that the variance should *increase* as you add more and more random objects together. The calculation we obtain in Example 6, where we found $\sigma_S^2 = 2\sigma_Z^2$, certainly suggests this.

But when we compute the average A , we're not summing together the Z_i s, there is also the factor $\frac{1}{n}$. Here is another way to write the random variable A that will help us see what is going on:

$$A = \frac{Z_1}{n} + \frac{Z_2}{n} + \dots + \frac{Z_n}{n}.$$

The intuition that adding up together multiple independent random variable will result in a combined variance that is higher than the individual variances is correct. The reason the variance ends up being smaller is because we're adding together copies of $\frac{Z}{n}$, which has variance $\frac{1}{n^2} \cdot \sigma_Z^2$, so the effects of summing together n copies is countered by the factor $\frac{1}{n^2}$.

The above observations about the mean and variance of the random variable A in the i.i.d. case are the basis of the *central limit theorem*, which is an important theoretical result we'll learn about in Section 2.8. Spoiler: the above equations for μ_A and σ_A^2 are approximately true for the average of *any* random variable X .

2.5.7 Discussion

We briefly touch upon the topic of “bulk” and “tails” of a distribution, since this type of reasoning and calculations will come up *a lot* in the rest of the book.

Bulk of the normal distribution

We're often interested in calculating an interval that contains “the bulk” of the distribution f_X . We want to find an interval (a subset of the sample space) where most of the observations of the random variable X will fall. One way to construct such a high-density interval is to define it in terms of the mean μ_X and the standard deviation σ_X of the random variable. For example, we can define the interval I_k that contains all values that are within k standard deviations of the mean of the random variable X as follows:

$$I_k = [\mu_X - k\sigma_X, \mu_X + k\sigma_X].$$

We can then compute the probability $\Pr(X \in I_k)$ (how likely is X to be within k standard deviations of its mean), for different values of the parameter k .

For any normally distributed random variable $N \sim \mathcal{N}(\mu_N, \sigma_N)$ like the random variable `rvN` that we saw in Example 2, we can calculate the value of the probability $\Pr(N \in I_k)$ numerically:

$$\Pr(N \in I_k) = \Pr(\{\mu_N - k\sigma_N \leq N \leq \mu_N + k\sigma_N\}) = p_k,$$

where $p_k = 0.682$ for $k = 1$, $p_k = 0.954$ for $k = 2$, and $p_k = 0.997$ for $k = 3$. See Figure 2.68 for an illustration. These total-probability-within- k -standard-deviations-of-the-mean values are the same for all normally distributed variables, and is sometimes called the “68-95-99.7 rule.” The interval I_1 is not particularly interesting, since it

covers only 68.2% of the probability mass of the random variable X . The interval I_2 contains 95.4% of the total weight of the distribution, which is “most” of it. Intuitively, if we predict the outcome $\{N \in I_2\}$ we’ll be correct 95.4% of the time.

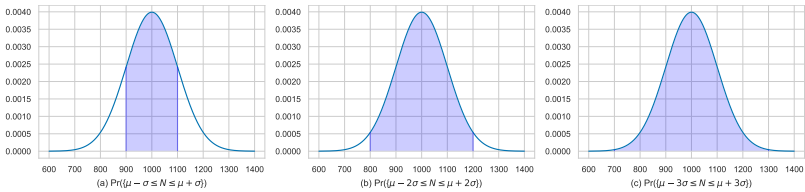


Figure 2.68: The probability of observing the random variable N within k standard deviations of the mean. In (a) we show the calculation $\Pr(\{900 \leq N \leq 1100\}) = 0.682$, which is roughly two thirds of the total mass of the distribution. The shaded region in (b) show the calculation $\Pr(\{800 \leq N \leq 1200\}) = 0.954$, which means 95.4% of the probability mass of f_N is situated within two standard deviations of the mean. The shaded region in (c) shows the three-sigma confidence interval that contains 99.7% of the probability.

The technical term for the interval I_k is *confidence interval*, meaning we’re confident, to a certain degree of probability p_k , that future outcomes of the random variable N will fall in this interval. We say $I_2 = [\mu_N - 2\sigma_N, \mu_N + 2\sigma_N] = [800, 1200]$ is a 95.4% confidence interval for the random variable N . This means, if we generate millions of observations from the random variable N , in the long run, 95.4% of these observations will be contained in the interval $[\mu_N - 2\sigma_N, \mu_N + 2\sigma_N]$. For an even higher degree of “confidence,” we can choose the three-sigma interval $I_3 = [\mu_N - 3\sigma_N, \mu_N + 3\sigma_N] = [700, 1300]$, which contains 99.7% of all observations.

Confidence intervals play an important role in statistics, whenever we estimate some quantity, we’ll report a confidence interval.

Tails of the normal distribution

The “tails” of the distribution contain the unlikely outcomes for the random variable. Figure 2.69 shows the tails of the distribution f_N , which are defined as the observations that are more than k standard deviations away from the mean.

The observations in the tails of the distribution are deemed “surprising” and “unexpected.” The notion of “unexpected outcome” plays a central role in the statistical concept of hypothesis testing (Section 3.X), in the next chapter.

If we observe an outcome that is part of the tails of some distribution, we’ll label it “unexpected” and interpret this result as interesting (statistically significant), and worthy of reporting and

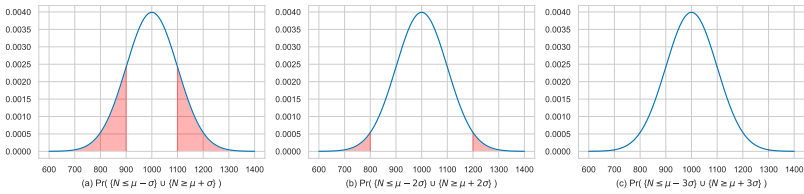


Figure 2.69: Illustration of the tails of the distribution f_N for the random variable $N \sim \mathcal{N}(1000, 100)$. The highlighted regions correspond to the complements of the regions highlighted in Figure 2.68. Subplot (b) shows that the probability of observing values of N that are more than two standard deviations away from the mean μ_N is 4.55%, which we can qualify as “very unlikely.” Subplot (c) shows the probability of observing N more than three standard deviations away from its mean is 0.27%, which we can qualify as “extremely unlikely.”

publication in scientific journals. The whole hypothesis testing procedure can be summarized as a “is it in the tail of the distribution” check, where the distribution in question is specially designed to model the probability of different outcomes under the current theory. If the observed outcome is “unexpected” under the current theory, then this lends support to the need to seek alternative theories.

At the risk of repeating myself, I’ll remind you the above calculations apply to *all* normally distributed random variables. In other words, for any random variable $X \sim \mathcal{N}(\mu_X, \sigma_X)$, the probability $\Pr(X \in [\mu_X - 2\sigma_X, \mu_X + 2\sigma_X])$ is 95.4%. This is a really powerful idea: we’ve obtained a result that applies to *all* normally distributed random variables, which are used to model many kinds of real-world situations. If we can calculate the mean μ_X and the standard deviation σ_X of this random variable, then we can compute a 95.4% confidence interval for it based on the two-standard-deviations-away-from-the-mean formula.

Exercises

TODO: select cont. RVs problems to convert to exercises

TODO: other uniform;

TODO: another normal calc.

Links

[Comprehensive list of hundreds of probability models]

https://en.wikipedia.org/wiki/List_of_probability_distributions

[Probability theory chapter from a book by Simon Hubbert]

https://bookdown.org/S_hubbert/mathematics_of_financial_derivatives/Prob-Th.html

[Video explainer about the moments of a distribution]

<https://www.youtube.com/watch?v=fv5QB3eK7jA>

2.6 Inventory of continuous distributions

We'll now continue the work we started in Section 2.3 to complete the list of the most important probability distribution used in statistics. In the previous section, we already saw some examples involving the uniform distribution $\mathcal{U}(\alpha, \beta)$ and the normal distribution $\mathcal{N}(\mu, \sigma)$, but there are several other continuous distributions that you need to know about.

Recall the notation $X \sim \mathcal{M}(\theta)$ which describes the random variable X distributed according to model \mathcal{M} with parameters θ . In this section, we'll learn about the different probability models \mathcal{M} that are available, and different choices of their parameters θ . Specifically, we'll talk about the following continuous probability distributions:

- Uniform $\mathcal{U}(\alpha, \beta)$: assign constant probability density over the interval $[\alpha, \beta]$.
- Exponential $\text{Expon}(\lambda)$: decaying probability curve defined for nonnegative real numbers.
- Normal $\mathcal{N}(\mu, \sigma)$: the normal or Gaussian distribution with mean μ and standard deviation σ .
- Standard normal $\mathcal{N}(0, 1)$: a special case of the normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$.
- Student's t -distribution: resembles the standard normal, but has "heavy" tails.
- Snedecor's F distribution: related to the ratio of variances.
- Chi-square χ^2 : related to the sum of squared deviations.

We'll also briefly mention the gamma distribution $\text{Gamma}(\alpha, \lambda)$, the beta distribution $\text{Beta}(\alpha, \beta)$, and the Cauchy distribution $\text{Cauchy}(x_0, \gamma)$, but you should consider these as "optional" reading material.

Figure 2.70 shows six examples, of the most common continuous probability distributions. Note the variety of "shapes" of the probability density functions. Compare the shapes in Figure 2.70 with the shapes of the discrete distributions that we saw earlier in Figure 2.27.

2.6.1 Math prerequisites

Let's quickly introduce some math concepts that we'll need later on in this section. Recall factorial function $n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$, which we used a lot in calculations for discrete probability distributions. The *gamma* function is a generalization of the factorial function to the space of continuous variables, and will similarly play a big role in the definitions of continuous random variables.

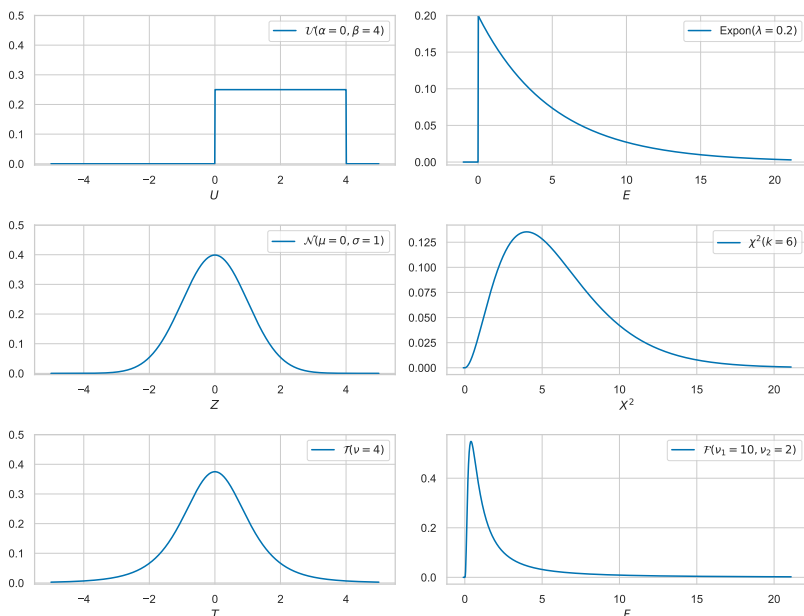


Figure 2.70: Probability density plots of 6 common probability distributions.

The gamma function The gamma function $\Gamma(z)$, denoted using the capital Greek letter *gamma*, is defined as the following complicated-looking integral:

$$\Gamma(z) \stackrel{\text{def}}{=} \int_0^{\infty} t^{z-1} e^{-t} dt.$$

You'll rarely need to compute values of the gamma function by hand, and instead use a computer (see code example below). For now, let's start by looking at the graph of the function $\Gamma(z)$ shown in Figure 2.71.

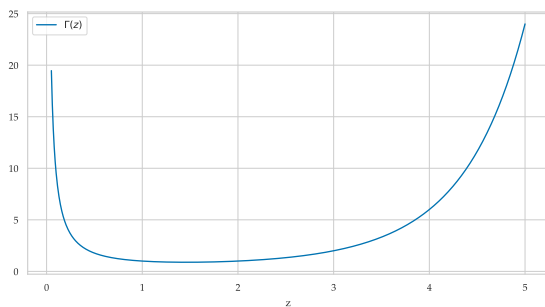


Figure 2.71: Graph of the gamma function Γ on the interval $(0, 5]$.

The gamma function $\Gamma(z)$ is defined for all $z > 0$, and it obeys the

following recursive relationship when evaluated at integer inputs:

$$\Gamma(z + 1) = z \cdot \Gamma(z),$$

which is similar to the structure obeyed by the factorial function. Indeed, the gamma function evaluated at $n + 1$ is equal to the factorial function $n!$:

$$\Gamma(n + 1) = n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1.$$

If you need to compute the value of $\Gamma(z)$, you can rely on the Python function `gamma` defined in the `scipy.special` module.

```
>>> from scipy.special import gamma as gammaf
>>> gammaf(5)    # = 4! = 4*3*2*1
24.0
```

code
2.6.1

In the above code example, we imported the function `scipy.special.gamma` under the alias `gammaf`. We do this to avoid any possible confusion with the gamma distribution `scipy.stats.gamma`, which has the same name.

The value of $\Gamma(z)$ varies smoothly between integers inputs. We can observe this by evaluating $\Gamma(z)$ for a sequence of inputs ranging between 4 and 5:

```
>>> [gammaf(z) for z in [4, 4.1, 4.5, 4.9, 5]]
[6.0, 6.81, 11.63, 20.67, 24.0]
```

code
2.6.2

See Figure 2.71 for the graph of the function. Use the graph to visually confirm the coordinate pairs $(z, \Gamma(z))$ from code block 2.6.2 all lie on the graph of the function $\Gamma(z)$.

You don't need to worry about the gamma function too much. I know it sounds and looks complicated, but you're not expected to memorize its formula, or ever have to do math calculations with it. I'm only introducing the Γ function here, so you won't be like, "what the hell is this Γ thing!?", when you see it later on in one of the formulas in this section. Don't freak out, and remember that $\Gamma(z)$ as just fancy math notation that behaves like the factorial $(z - 1)!$.

2.6.2 Continuous distributions reference

We'll now switch to "reference mode" and show examples of the six most common families of continuous probability distributions: uniform, exponential, normal, Student's t -distribution, Snedecor's F distribution, and the Chi-squared distribution. For each of these distributions, we'll show the definitions and formulas, describe their properties, and provide a minimal code example. We'll also include additional links for further study.

All of these distributions are used in one way or another in statistics, so it's worth spending some time to get to know them. Remember that you're not supposed to memorize any of the formulas and definitions in the next MM pages. You just need to know that these distributions exist, so you can refer back to this section if you need to look up facts about them. You want to focus on "visual understanding" by looking at the graphs of their probability density functions, and remember the "story" behind the distribution, but no need to memorize formulas.

OK enough preliminaries! Let's start the inventorying of continuous distributions. Ready? Let's go!

Uniform

The continuous uniform distribution $\mathcal{U}(\alpha, \beta)$ assigns equal probability to all numbers on the interval $[\alpha, \beta] = \{\alpha \leq X \leq \beta\}$. The probability density function is

$$f_X(x) = \begin{cases} \frac{1}{\beta - \alpha} & \text{for } \alpha \leq x \leq \beta, \\ 0 & \text{for } x < \alpha \text{ or } x > \beta. \end{cases}$$

In words, we see each x between α and β is equally likely to occur, and values of x outside this range have zero probability of occurring.

Figure 2.72 shows various uniform distributions defined over different intervals $[\alpha, \beta]$. Note the longer the length of the interval $\beta - \alpha$, the lower the probability density becomes, so that the total area under the curve remains 1.

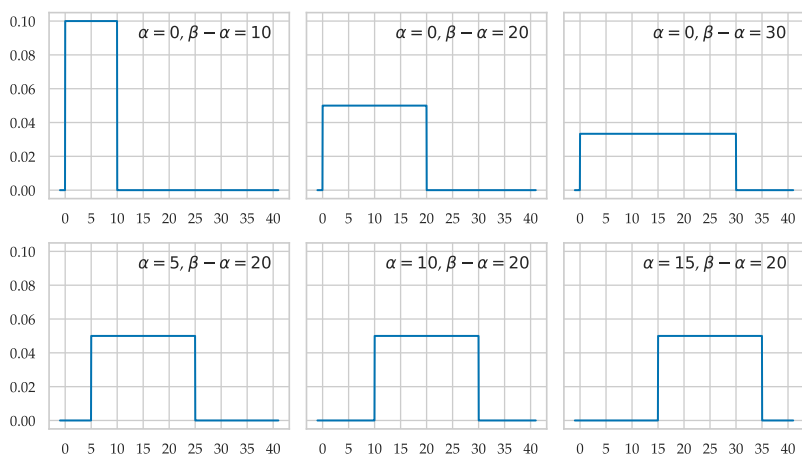


Figure 2.72: Plot of uniform distributions for various choices of the parameter α and β .

Its mean and variance are

$$\mu = \frac{\alpha + \beta}{2} \quad \text{and} \quad \sigma^2 = \frac{(\beta - \alpha)^2}{12}.$$

You'll be asked to verify these formulas in exercise E2.39 and problem P2.1.

Cumulative distribution function The cumulative distribution function of the uniform distribution looks like a straight line that increases from 0 to 1 as the input b varies from α to β . Figure 2.73

shows a side-by-side plots of the probability density function (pmf) f_X and the cumulative distribution function (CDF) F_X for the uniform random variable $X \sim \mathcal{U}(2, 7)$. Recall the CDF is obtained from the pdf using integration:

$$F_X(b) = \Pr(\{X \leq b\}) = \int_{-\infty}^b f_X(x).$$

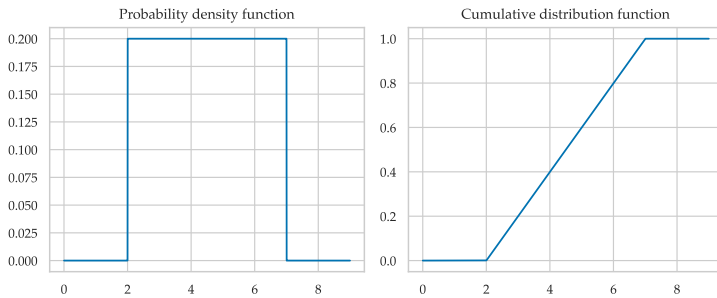


Figure 2.73: Graphs of the probability density function f_X and the cumulative density function F_X for the random variable $X \sim \mathcal{U}(2, 7)$.

Computer model We can use the uniform model from `scipy.stats` to create computer models uniformly distributed random variables. For example, we can create the computer from the random variable $X \sim \mathcal{U}(2, 7)$ using the following code:

```
>>> from scipy.stats import uniform
>>> alpha = 2
>>> beta = 7
>>> rvU = uniform(alpha, beta-alpha)
```

code
2.6.3

Note when initializing the uniform model, we pass in the distance $\beta - \alpha$ as the second parameter, and not β as in the math notation.

To generate 10 random observations from the random variable `rvU`, we call the `.rvs()` method, as shown below.

```
>>> rvU.rvs(10)
array([2.69746, 3.460723, 3.831809, 4.280349, 5.925879,
       2.99836, 4.571172, 4.962072, 2.232252, 5.037724])
```

code
2.6.4

Observe all the numbers are between $\alpha = 2$ and $\beta = 7$, which are the initialization parameters we specified when we created the random variable `rvU`.

See Figure 2.73 for the graphs of the pdf and CDF of the random variable `rvU`.

The standard uniform The *standard* uniform distribution is a special case of the uniform distribution over the interval $[0, 1]$. The standard uniform is important because it allows us to generate other types of random variables thanks to the inverse-CDF trick, which we'll discuss further in Section 2.7.

Most programming languages provide functionality for generating random observations according to the standard uniform $\mathcal{U}(0, 1)$. Python has at least three options for generating random numbers uniformly distributed between 0 and 1.

The first option, which we saw above, is to create the random variable `rvU = uniform(0, 1)` and then call the method `rvU.rvs(n)`.

The second option is to import the Python module `random` and use the function `random.random()` as shown below.

```
>>> import random
>>> random.random()
0.23796462709189137
```

code
2.6.5

Every time you call the function `random.random()`, the computer will generate a new random number from the interval $[0, 1]$.

The third option for generating numbers from the distribution $\mathcal{U}(0, 1)$ is to call the function `random.rand()` which is part of the NumPy package.

```
>>> import numpy as np
>>> np.random.rand()
0.3745401188473625
```

code
2.6.6

Calling `np.random.rand(n)` will return a NumPy array of n random numbers drawn from $\mathcal{U}(0, 1)$.

Similarly named functions are available in other programming languages and computational platforms. In Excel, you can use the function `RAND()` to generate a random number between 0 and 1. The equivalent function in R is `runif(1)`, which is short for random uniform and the number indicates we just want one draw.

Applications We can use the uniform random variable to generate random variables from other distributions. For example, suppose we want to generate observations of a coin toss random variable which comes out heads 50% of the time and tails 50% of the time. We can use the standard uniform random variables obtained from `random.random()` and split the outcomes at the “halfway point” of the sample space, to generate the 50-50 randomness of a coin toss. The Python function `flip_coin` shows how to do this:

```
>>> def flip_coin():
    u = random.random() # random number in [0,1]
    if u < 0.5:
        return "heads"
```

code
2.6.7

```
else:
    return "tails"
```

To generate the outcome of a random coin toss, simply call the function `flip_coin`:

```
>>> flip_coin()
'heads'
```

code
2.6.8

To generate a list of 10 outcomes of flipping the coin, write a for loop that calls the function 10 times.

```
>>> [flip_coin() for i in range(0,10)]
['tails', 'tails', 'heads', 'heads', 'tails',
 'heads', 'heads', 'tails', 'heads', 'tails']
```

code
2.6.9

We'll continue the discussion on generation of random variables in ... TODO: setup a placeholder so we can reference it.

Relations to other distributions

- The standard uniform $\mathcal{U}(\alpha = 0, \beta = 1)$ is equivalent to the beta distribution with $\text{Beta}(1, 1)$.
-

[Continuous uniform distribution on Wikipedia]

https://en.wikipedia.org/wiki/Continuous_uniform_distribution

Exponential

The exponential distribution $X \sim \text{Expon}(\lambda)$ describes the “waiting time” between two events, when the base rate is λ . The sample space of this random variable is $\mathcal{X} = [0, \infty)$. The probability density function is

$$f_X(x) = \lambda e^{-\lambda x}.$$

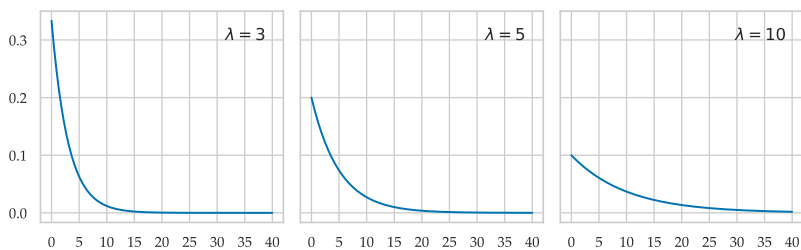


Figure 2.74: Plot of the probability density function of the exponential model for three different values of the parameter λ

The mean and variance of the exponential distribution are

$$\mu = \mathbb{E}_X[X] = \frac{1}{\lambda} \quad \text{and} \quad \sigma_X^2 = \mathbb{E}[(X - \mu_X)^2] = \frac{1}{\lambda^2}.$$

You’ll be asked to compute these formulas in exercises E2.41 and E2.42.

The cumulative distribution is easy to compute (see Exercise E2.40), and it is given by

$$F_X(b) = 1 - e^{-\lambda b}.$$

Memoryless property The exponential distribution is *memoryless*, meaning it has a self-similar shape for all values of x . The *memoryless property* is expressed as

$$\Pr(X > x) = \Pr(X > x + y | X > y).$$

In words, this means the probability that a success will take longer than x seconds starting at 0, is equal to the probability of a success taking $x + y$ seconds, given that no success happened in the first y seconds. In other words, the exponential random variable doesn’t become more likely to occur even after a long period where it hasn’t occurred. The probability of outcome of success in the next second stays constant, no matter what happened previous seconds, hence the name “memoryless” used to describe this property.

For example, in the waiting time before the next call X can be modelled as an exponential distribution then the memoryless property says that the probability of receiving a call in the next 30 seconds given that you have already waited five minutes is the same as during the first 30 seconds.

The exponential distribution is the only continuous probability distribution that has the memoryless property. The geometric distribution we saw in Section 2.3 is also memoryless.

Computer model We can use the `expon` model from `scipy.stats` to create computer models exponentially distributed random variables. For example, we can create the computer from the random variable $rvE \sim \text{Expon}(\lambda = 7)$ using the following code:

```
>>> from scipy.stats import expon
>>> loc = 0
>>> lam = 7
>>> scale = 1/lam
>>> rvE = expon(loc, scale)
```

code
2.6.10

To compute the mean and variance, we call use the methods on the random variable `rvE`:

```
>>> rvE.mean(), rvE.var()
(0.14285714285714285, 0.02040816326530612)
```

code
2.6.11

Alternatively, we can use the math formulas for μ_E and σ_E^2 , which give the same answer:

```
>>> 1/lam, 1/lam**2
(0.14285714285714285, 0.02040816326530612)
```

code
2.6.12

Applications Radioactive decay?

Relations to other distributions

- The Poisson distribution counts the number of events that will occur during some interval with base rate λ , while the exponential measures the time between events.
- A geometric random variable is the floor of an exponential random variable.
- The exponential distribution is a special case of the gamma distributions for with gamma shape $\alpha = 1$.
- The sum of n exponential random variables with parameter λ a gamma distribution with parameters $\alpha = n$ and $\lambda = \lambda$.

[Wikipedia page]

https://en.wikipedia.org/wiki/Exponential_distribution

Normal

The *normal distribution* $\mathcal{N}(\mu, \sigma)$ has the probability density function:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

where μ is the mean and σ is the standard deviation. We use the notation $\mathcal{N}(\mu, \sigma)$ to describe the distribution as math, and `norm(mu, sigma)` to describe as computer model.

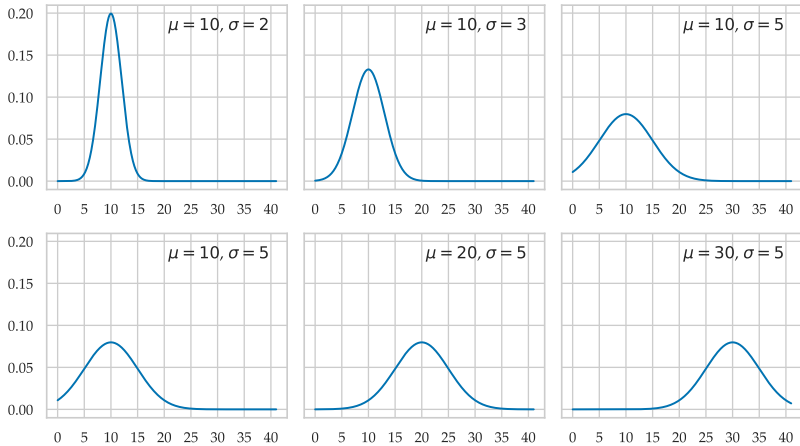


Figure 2.75: Plot of normal distributions for various choices of the parameters μ and σ .

The mean and variance of the distribution are as follows:

$$\mathbb{E}_X[X] = \mu \quad \mathbb{E}[(X - \mu_X)^2] = \sigma^2.$$

Computer model To create a normally distributed random variable $\text{rvN} \sim \mathcal{N}(\mu = 1000, \sigma = 100)$, we use the following code:

```
>>> from scipy.stats import norm
>>> mu = 10
>>> sigma = 3
>>> rvN = norm(mu, sigma)
```

code
2.6.13

Let's verify the formulas for the mean and variance:

```
>>> rvN.mean(), rvN.var()
(10.0, 9.0)
```

code
2.6.14

Applications Every quantity in this world that is computed as the sum of a large number of independent observations ends up looking like a normal distribution.

The normal distribution is used as a generic all-purpose distribution whenever we want to describe some unknown distribution with mean μ and standard deviation σ . (or if based on sample statistics \bar{x} and s too).

The normal distribution $\mathcal{N}(\mu = np, \sigma = \sqrt{np(1-p)})$ can be used as an approximation for the binomial distribution $\text{Binom}(n, p)$, when the sample size n is large ($n \geq 20$). This is known as the Moivre–Laplace approximation.

Relations to other distributions In addition to the normal approximation to the binomial distribution which we discussed above, the normal distribution is connected to many other probability distributions:

- The Poisson is also ...
- Related to Student's t-distribution when using estimated
- It is equal to the standard normal Z if we perform the “standardization” transformation $Z = \frac{X - \mu_X}{\sigma_X}$.
- The sum of n normally distributed random variables, is also normally distributed: TODO show CLT formula (without explaining). This is big thing = TODO: FWD reference to CLT

[The normal distribution]

https://en.wikipedia.org/wiki/Normal_distribution

Standard Normal

The *standard normal distribution* is denoted $Z \sim \mathcal{N}(0, 1)$ and it has the probability density function:

$$f_Z(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}.$$

Note this is the equation as the general normal distribution $\mathcal{N}(\mu, \sigma)$ if choose the parameters $\mu = 0$ and $\sigma = 1$. The term “standard” refers to this choice of 0 for the mean and 1 for the standard deviation. See Figure 2.76 for the graph of the probability density function f_Z .

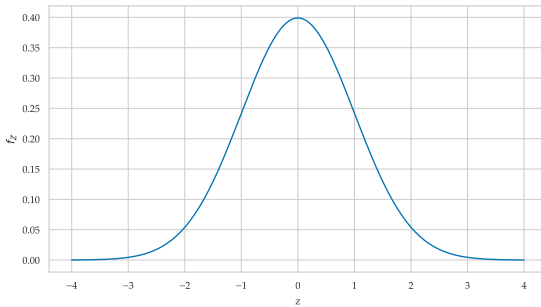


Figure 2.76: Plot of the probability density function f_Z for the standard normal Z .

The mean and variance of the distribution are as follows:

$$\mathbb{E}_Z[Z] = 0 \quad \mathbb{E}_Z[(Z - 0)^2] = 1.$$

Relation to the generic normal distribution All normal distributions have essentially the same “shape.” Consider the random variable X which is normally distributed with mean μ and standard deviation σ :

$$X \sim \mathcal{N}(\mu, \sigma).$$

The two random variables Z and X are related by the following equation:

$$Z = \frac{X - \mu}{\sigma},$$

which subtracts the mean and divides by the standard deviation. Every Gaussian random variable can be transformed to the standard normal distribution using this transformation.

For every calculation you might want to do with the random variable X , there is an equivalent calculation you can carry out using the random variable Z :

$$F_X(a) = \Pr(X \leq a) = \Pr\left(Z \leq \frac{a - \mu_X}{\sigma_X}\right) = F_Z\left(\frac{a - \mu_X}{\sigma_X}\right),$$

where $F_X(a) = \int_{-\infty}^a f_X(x) dx$ is the cumulative distribution function (CDF) of the random variable X , and F_Z is the CDF of the standard normal. This means it suffices to know the values of the CDF for the standard normal distribution F_Z , where $Z \sim \mathcal{N}(0,1)$, and the calculations for all other normal distributions can be obtained after a suitable transformation.

Probability calculations The function $F_Z : \mathbb{R} \rightarrow [0,1]$ can be used in two directions. Either we have a given value of z and we want to calculate $F_Z(z)$ (the cumulative probability of the random variable Z taking on this or any smaller value), or we start with some probability value q and want to compute the corresponding z -value z_q such that $F(z_q) = q$, in other words $z_q \equiv F^{-1}(q)$.

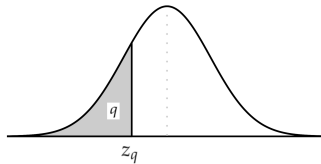


Figure 2.77: Illustration of the cumulative probability density calculations of the standard normal. The value z_q is such that a total probability of q is enclosed in the left tail of the distribution.

Computer model To create a

```
>>> from scipy.stats import norm
>>> rvZ = norm(0, 1)
```

code
2.6.15

Let's verify the mean and variance of the random variable `rvZ` are as specified.

```
>>> rvZ.mean(), rvZ.var()
(0.0, 1.0)
```

code
2.6.16

Calculations using the cumulative distribution function Several statistics procedures involve calculations based on the cumulative distribution function F_Z , so it is a good idea to get to know them well.

We'll start with some example questions that we can answer using the cumulative distribution F_Z . Suppose you observed the value $z_0 = -2.3$, and you want to know the probability of observing the value z_0 or more negative for the random variable Z . You can compute the answer using the $F_Z(-2.3) = \Pr(\{Z \leq -2.3\}) = \text{rvZ.cdf}(-2.3)$, as shown below.

code
2.6.17

```
>>> rvZ.cdf(-2.3)
0.010724110021675809
```

This is called a “left tail” calculation because the calculation involves integrating the left tail of the normal distribution until $z_0 = -2.3$.

You can obtain the symmetric calculation of the right tail using one-minus the cumulative distribution:

```
>>> 1 - rvZ.cdf(2.3)
0.010724110021675837
```

code
2.6.18

This corresponds to the probability of observing value $z_0 = 2.3$ or greater: $\Pr(\{Z \geq 2.3\})$.

The more general calculation you’ll see in the statistics chapter, is to compute the probability of “observing z_0 or a more extreme value,” which corresponds to the two-tailed probability calculation $\Pr(\{|Z| \geq |z_0|\}) = \Pr(\{Z \leq -2.3\}) + \Pr(\{Z \geq 2.3\})$, which we can compute by adding together the contribution from the left tail and right tail we obtained above:

```
>>> rvZ.cdf(-2.3) + (1-rvZ.cdf(2.3))
0.021448220043351646
```

code
2.6.19

Calculations using the inverse CDF The inverse cumulative distribution function of the standard normal F_Z^{-1} will also come up often in the statistics chapter. In the inverse direction, we’re starting from some proportion $q \in [0, 1]$ of the total probability, and we want to know the smallest value z_q such that $\Pr(\{Z \leq z_q\}) = q$.

For example, we can find the cutoff value of the 5% left tail of the standard normal distribution using

```
>>> rvZ.ppf(0.05)
-1.6448536269514729
```

code
2.6.20

This tells us $\Pr(\{Z \leq -1.64\}) = 0.05$.

To find the cutoff value of the 5% right tail of the distribution, we use the code

```
>>> rvZ.ppf(0.95)
1.6448536269514722
```

code
2.6.21

This tells us $\Pr(\{Z \leq 1.64\}) = 0.95$ and therefore $\Pr(\{Z \geq 1.64\}) = 0.05$.

The complement of the above two regions contains the middle 90% of the distribution: $\Pr(\{-1.64 \leq Z \leq 1.64\}) = 0.90$. This is called the 90% *confidence interval* for the random variable Z , which means 90% of observations from the standard normal will fall in that interval. We can also use the method `rvZ.interval()` to compute the confidence interval in one step:

code
2.6.22

```
>>> rvZ.interval(0.9)
(-1.6448536269514729, 1.6448536269514722)
```

Applications The standard normal distribution is one of the top-three distributions used in statistics procedures. It is used for all statistical tests where the variance of the distribution is known, and for statistical tests that compare two proportions (based on the normal approximation to binomial distribution).

The standard normal is also used to compute the z-score of any value x in a dataset: $z_x = \frac{x - \text{Mean}}{\text{Std}}$. The z-score can then be used to identify “outliers,” that is values that are exceptional or unexpected. For example, one common approach for detecting outliers is to look for values whose z-score is $|z_x| \geq 3$. These values are more than three standard deviations away from the mean.

Relations to other distributions

- We can simulate general normal $X \sim \mathcal{N}(\mu, \sigma)$ using the scale-and-location transformation $X = \mu + \sigma Z$.
- The general normal $X \sim \mathcal{N}(\mu, \sigma)$ is the same as Z after the standardization transformation $Z = \frac{X - \mu}{\sigma}$.
-

[The standard normal defined on Wikipedia]

https://en.wikipedia.org/wiki/Normal_distribution#Standard_normal_distribution

Student's t -distribution

The shape of the t -distribution is similar to the shape of the normal, but with “heavier” tails. We’ll use Student’s t -distribution for several statistical analysis procedures in situations when we need to estimate the population variance based on the sample variance.

There is a whole family of t -distributions defined by the different values of the parameter ν , which represents the *number of degrees of freedom* of the distribution, and determines the shape of the distribution. We’ll denote the degrees of freedom parameter using the Greek letter “ ν ” (pronounced like “new”) in math equations, and “`df`” in code examples.

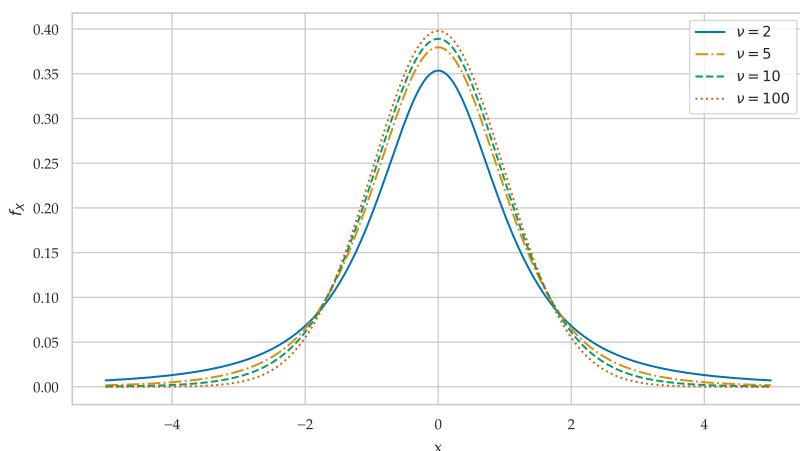


Figure 2.78: Plot of the probability densities of Student’s t -distribution for four choices of the degrees of freedom ν . For large values of ν , the shape of Student’s t -distribution becomes equal to the standard normal $\mathcal{N}(0, 1)$.

The probability density function of Student’s t -distribution is a really scary-looking math expression, which I’m about to show you. Rest assured, you’ll never need to compute this function by hand—we’re only showing it for completeness. The probability density function for Student’s t -distribution with ν degrees of freedom is

$$f_X(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}.$$

I warned you the formula will look complicated!

The mean and variance of Student's t -distribution are

$$\mu_X = 0 \quad \text{and} \quad \sigma_X^2 = \frac{\nu}{\nu - 2}, \text{ for } \nu > 2.$$

Historical background Student's t -distribution has an interesting history. The statistician William Gosset came up with this distribution while working at the Guinness brewing company, but because he was in “industry” he published under the pseudonym “Student,” and the name was never changed. Presumably, he used a pseudonym so readers would not be able to tie the analysis to Guinness brewing. Imagine if Irish people hear there are “bad batches” of Guinness that the factory has to throw out—people would storm the factory for sure asking to be given the bad batches!

Computer model To create a random variable $\widehat{rv}T$ with probability distribution $\mathcal{T}(\nu = 10)$, we import the `t` model from `scipy.stats` and initialize it with degrees of freedom parameter.

```
>>> from scipy.stats import chi2
>>> df = 10
>>> rvT = t(df)
```

code
2.6.23

The mean and the variance of the distribution are:

```
>>> rvT.mean(), rvT.var()
(0.0, 1.25)
```

code
2.6.24

You can verify the value of the variance matches the math formulas we showed above. Note the variance is slightly larger than the standard normal Z , which has variance $\sigma_Z^2 = 1$.

Calculations using the cumulative distribution function We can then use the method `rvT.cdf(b)` to obtain values of the cumulative distribution function $F_T(b)$ and the method `rvT.ppf(q)` to obtain values of the inverse CDF $F_T^{-1}(q)$.

For example, the if we observe the value $t_o = -2.3$, we can calculate the probability of observing t_o or more negative for the random variable T using the cumulative distribution function $F_T(-2.3) = \Pr(\{T \leq -2.3\}) = \text{rvZ.cdf}(-2.3)$, which has the value:

code
2.6.25

```
>>> rvT.cdf(-2.3)
0.022127156642143552
```

The probability $\Pr(\{T \leq -2.3\}) = 2.21\%$. Compare this with the value $\Pr(\{Z \leq -2.3\}) = 1.07\%$, which we obtained for the standard normal distribution. This is why we say the t -distribution has “heavy tails.”

Let’s now look at some calculations, based on the inverse cumulative distribution function $F_T^{-1}(q) = t_q$. We can obtain the 90% confidence interval for the t -distribution using the following code:

```
>>> rvT.ppf(0.05), rvT.ppf(0.95)
(-1.8124611228107341, 1.8124611228107335)
code
2.6.26
```

Compare this with the 90% interval for the standard normal: $[F_Z(0.05), F_Z(0.95)] = [-1.645, 1.645]$. Again, we see the t -distribution is more spread out as compared to the standard normal.

Cumulative distribution calculations Calculating values of the cumulative distribution function $F_X(b)$ of Student’s t -distribution and its inverse $F_X^{-1}(q)$ is a very common task in statistics, so let’s review visually the meaning of the probability calculations $F_X(b)$ and $F_X^{-1}(q)$.

The cumulative distribution function computes the integral of the probability density up to $x = b$:

$$F_X(b) = \Pr(\{T \leq b\}) = \int_{-\infty}^b f_X(x)$$

We’ll often be interested in the probability of the complementary event $\Pr(\{T \geq b\})$, which we can obtain as one minus the value of the value of the cumulative distribution

$$\Pr(\{T \geq b\}) = 1 - F_X(b).$$

For example, we want to compute the probability of observing the value of the t random variable equal to or greater than 2, you can compute this using $\Pr(\{T \geq 2\}) = 1 - F_X(2)$, which corresponds to the code `1 - rvT.cdf(2)`, where `rvT` is the probability model of the t distribution.

Suppose instead we’re interested in the inverse problem: we want to find the value t_q such that $F_X(t_q)$ is equal to q . Visually speaking, see Figure 2.79, we must choose the value t_q such that the total probability of values of t or smaller is equal to q . The inverse of the cumulative distribution function F_X^{-1} is defined precisely for this purpose: $t_q = F_X^{-1}(q)$.

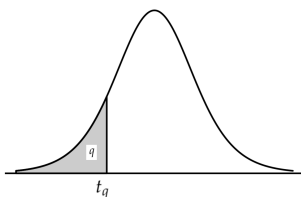


Figure 2.79: Illustration of the cumulative probability density calculations of the t distribution with ν degrees of freedom. The value t_q is such that a total probability of q is enclosed in the left tail of the distribution.

For example, if you want to find the cutoff value t_ℓ such that 0.05 (5 percent) of the distribution to the left of it, you can compute $t_\ell = F_X^{-1}(0.05)$ using the code `rvT.ppf(0.05)`. We can also find the cutoff value t_r such that 0.05 (5 percent) of the distribution to the right of it, compute $t_r = F_X^{-1}(0.95) = \text{rvT.ppf}(0.95)$. Note the interval $[t_\ell, t_r]$ contains 90% percent of the probability of the random variable T . In words, 90% of random observations from the random variable T will fall in the interval $[t_\ell, t_r]$, which means this is a 90% confidence interval, denoted $CI_{0.9}$.

Obtaining a confidence interval that contains $(1 - \alpha)$ of the total probability is described by the following general formula:

$$CI_{1-\alpha} = \left[F_X^{-1}(\alpha/2), F_X^{-1}(1 - \alpha/2) \right].$$

TODO: visual for CI & tails — back reference to bulk and tails discussion earlier in Section 2.1

TODO: mention 70% of STATS101 homework questions will involve this CI construction, or calculation $1 - F_T(t)$ to get p-value. this is how important the t-distribution is, as we'll explain in the next section.

Applications (1) Interpreting the values of the t -statistic: $t = \frac{\hat{\theta} - \mu_\theta}{\widehat{\text{se}}_{\hat{\theta}}}$, where μ_θ is the mean of an estimator and $\widehat{\text{se}}_{\hat{\theta}}$ is an estimated standard error.

Student's t distribution is used in conjunction with the t -statistic to perform statistical analysis in case where the population variance is estimated from a sample.

TODO EXAMPLE: calculate the probability of observing the test statistic $t = 5$ for the sample mean \bar{x} computed from samples of size $n = 9$ taken from a population with mean μ and variance σ^2 . (this is setting up the eprices analysis that is comping up in hypothesis testing...)

(2) Calculating CIs using the inverse-CDF of the t -distribution:

Relations to other distributions

- As ν goes to infinity, Student's t -distribution becomes the standard normal distribution.
- Related to the F -distribution $\mathcal{F}(1, \nu) = (\mathcal{T}(\nu)^2)$.

[Wikipedia page]

https://en.wikipedia.org/wiki/Student's_t-distribution

[The process of fattening the tails is called Studentization]

<https://en.wikipedia.org/wiki/Studentization>

Snedecor's F distribution

Snedecor's F -distribution is also called Fisher, or Fisher–Snedecor distribution, or sometimes also called the variance ratio distribution.

The probability density function is

$$f_X(x) \stackrel{\text{def}}{=} \frac{\Gamma(\frac{\nu_1 + \nu_2}{2})}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})} \left(\frac{\nu_1}{\nu_2}\right)^{\nu_1/2} \frac{x^{(\nu_1-2)/2}}{\left(1 + \frac{\nu_1}{\nu_2}x\right)^{(\nu_1 + \nu_2)/2}},$$

with sample space $[0, \infty)$.

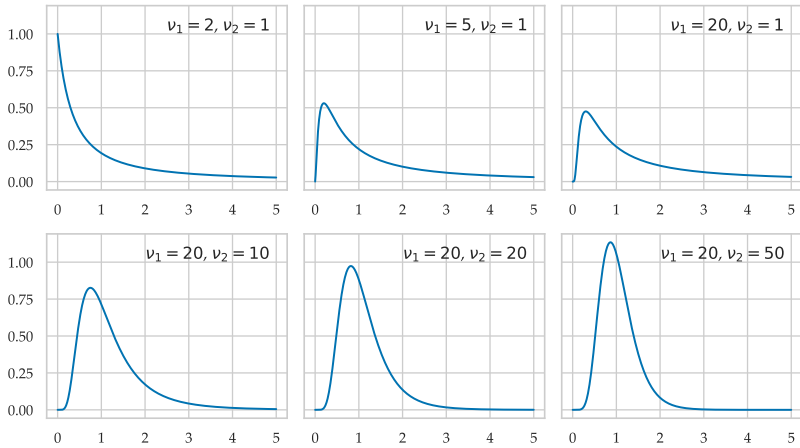


Figure 2.80: Plot of the probability densities of Snedecor's F -distribution for different choices of the parameters ν_1 and ν_2 .

The mean and variance of the distribution are as follows:

$$\mu_X = \frac{\nu_2}{\nu_1 + \nu_2}, \nu_2 > 2, \quad \sigma_X^2 = 2 \left(\frac{\nu_2}{\nu_2 - 2} \right)^2 \frac{\nu_1 + \nu_2 - 2}{\nu_1(\nu_2 - 4)}, \nu_2 > 4.$$

Computer model The code below shows how to create an instance of the F -distribution with degrees of freedom parameters $\nu_1 = 15$ and $\nu_2 = 10$.

```
>>> from scipy.stats import f
>>> df1, df2 = 15, 10
>>> rvF = f(df1, df2)
```

code
2.6.27

The mean and variance of the random variable `rvF` are

```
>>> rvF.mean(), rvF.var()
(1.25, 0.7986111111111112)
```

code
2.6.28

Applications Used in certain statistical tests...

Relations to other distributions

- Related to the chi-squared distribution $\mathcal{F}(\nu_1, \nu_2) = \frac{\chi^2(\nu_1)/\nu_1}{\chi^2(\nu_2)/\nu_2}$
- Related to the square of the t -distribution $\mathcal{F}(1, \nu) = (\mathcal{T}(\nu))^2$.

[Wikipedia page]

<https://en.wikipedia.org/wiki/F-distribution>

Chi-squared distribution

The χ^2 distribution is used in several statistics procedures. The superscript 2 gives us a hint that the quantity has something to do with squares. The Greek letter χ is spelled “chi” (rhymes with “bye”), so “ χ^2 ” is read “chi squared.”

Sample space is all the non-negative real numbers $[0, \infty) = \mathbb{R}_+$. The probability density of the χ^2 distribution with k degrees of freedom is

$$f_X(x) = \frac{1}{2^{\frac{k}{2}} \Gamma(k/2)} x^{\frac{k}{2}-1} e^{-\frac{x}{2}},$$

for $k \in \{1, 2, 3, \dots\}$.

The *degrees of freedom* parameter k determines the shape of the distribution. Figure 2.81 illustrates several plots for different values of k .

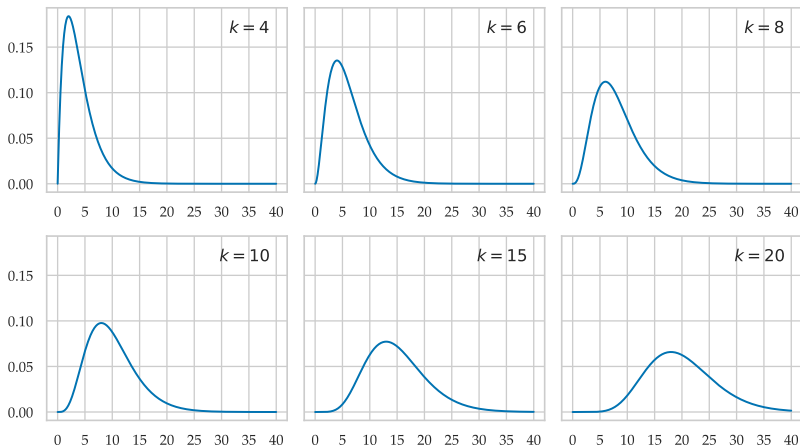


Figure 2.81: Plot of the Chi-squared distribution for different choices of the parameter k . As k gets larger, the peak of the distribution moves to the right.

The mean and variance of χ^2 -distribution with k degrees of freedom are as follows:

$$\mathbb{E}_X[X] = k \quad \mathbb{E}[(X - \mu_X)^2] = 2k.$$

Computer model Let’s create a χ^2 -distribution with $k = 10$ degrees of freedom.

```
>>> from scipy.stats import chi2
>>> k = 10
>>> rvX2 = chi2(k)
```

code
2.6.29

The mean and the variance of the random variable rvX2 are given by:

```
>>> rvX2.mean(), rvX2.var()
(10.0, 20.0)
```

code
2.6.30

Cumulative distribution calculations CDF and inverse CDF will be used a lot in statistical procedures, so let's review visually ...



Figure 2.82: Illustration of the probability calculations for χ^2 distribution with k degrees of freedom. The value $\chi^2_{q,k}$ is such that a total probability of q is enclosed in the left tail of the distribution.

For example, if we want to calculate the probability of observing a value greater than 20 for the χ^2 distribution with $k = 10$ degrees of freedom, we run the code:

```
>>> 1 - rvX2.cdf(20)
0.02925268807696113
```

code
2.6.31

The probability is 2.9%, which is fairly unlikely.

Applications The χ^2 distribution is the sampling distribution of the estimator that computes the sum of squares independent normally distributed random variables.

Used to obtain the sampling distribution of the sample variance estimator S^2 . If S^2 is the variance of a random sample of size n from a normal population having variance σ^2 , then the sampling distribution of $\frac{(n-1)S^2}{\sigma^2}$ is χ^2 with $n - 1$ degrees of freedom. We use this result for inferences concerning the population standard deviation σ .

Relations to other distributions

- χ^2 is related to sum of squares deviations of normally distributed random variables.
- Special case of the gamma function
- We can combine the standard normal Z and a $\chi^2(v)$ distribution to obtain Student's t -distribution: $T = \frac{Z}{\sqrt{\chi^2(v)/v}}$.

[Lots of useful info in the Wikipedia article]

https://en.wikipedia.org/wiki/Chi-squared_distribution

Gamma (optional)

The probability density function of the Gamma(α, λ) distribution is

$$f_X(x) \stackrel{\text{def}}{=} \frac{\lambda^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\lambda x}.$$

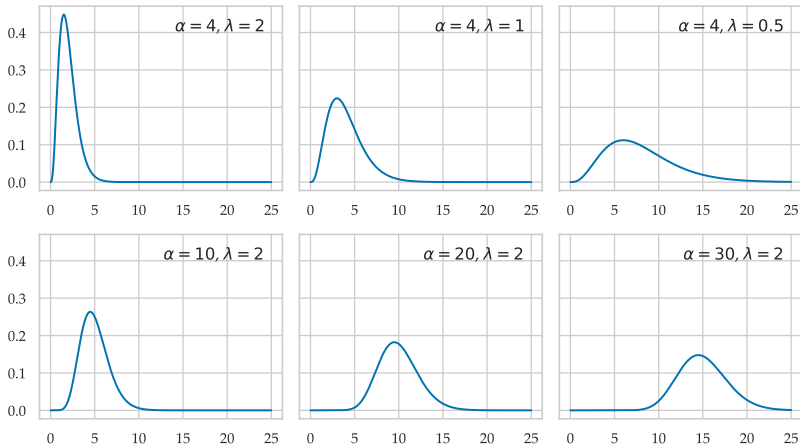


Figure 2.83: Plot of the gamma distribution for different choices of the parameters α and λ .

The mean and variance of the distribution are as follows:

$$\mathbb{E}_X[X] = \frac{\alpha}{\lambda} \quad \mathbb{E}[(X - \mu_X)^2] = \frac{\alpha}{\lambda^2}.$$

Computer model The code below shows how to create a gamma distribution with parameters $\alpha = 4$ and $\lambda = 2$.

```
>>> from scipy.stats import gamma as gammad
>>> alpha = 4
>>> loc = 0
>>> lam = 2
>>> beta = 1/lam
>>> rvG = gamma(alpha, loc, beta)
```

code
2.6.32

We import the gamma distribution under the alias `gammad` to avoid possible confusing with the gamma function, which has the same name. Note initializing the model expects a location parameter as the second argument, which in our case is 0, and the third argument is the scale parameter β which is defined as the inverse of λ : $\beta = \frac{1}{\lambda}$. The mean and the variance of the distribution are:

```
>>> rvG.mean(), rvG.var()
(2.0, 1.0)
```

code
2.6.33

Relations to other distributions

- The gamma distribution becomes the exponential distribution when $\alpha = 1$.
- The gamma distribution becomes the χ^2 distribution when $\alpha = \frac{k}{2}$ and $\beta = 2$.
- Related to Poisson via...
- The sum of independent gamma variables is also a gamma variable. If $X_1 \sim \text{Gamma}(\alpha_1, \lambda)$ and $X_2 \sim \text{Gamma}(\alpha_2, \lambda)$, then $X_1 + X_2 \sim \text{Gamma}(\alpha_1 + \alpha_2, \lambda)$

[Gamma function]

https://en.wikipedia.org/wiki/Gamma_function

[Gamma probability distribution]

https://en.wikipedia.org/wiki/Gamma_distribution

Beta (optional)

The Beta(α, β) distribution has the probability density function

$$f_X(x) = \frac{1}{\mathcal{B}(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1},$$

where $\mathcal{B}(\alpha, \beta)$ is the value of the beta function, which is defined in terms of the gamma function $\mathcal{B}(\alpha, \beta) \stackrel{\text{def}}{=} \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$.

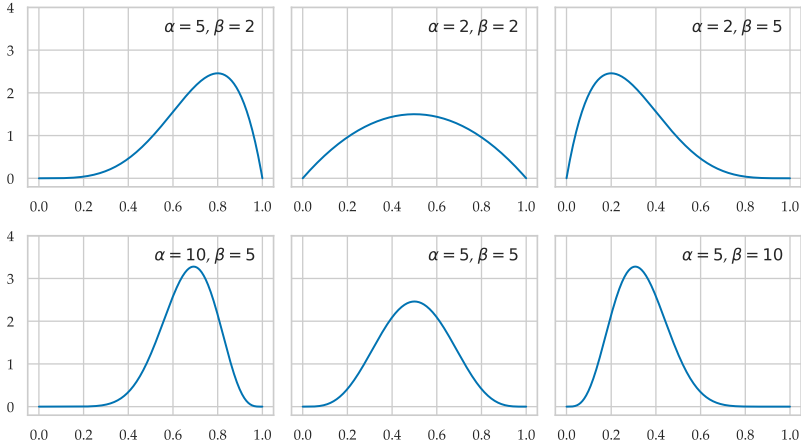


Figure 2.84: Plot of the beta distribution for different choices of the parameters α and β .

The mean and variance of the distribution are as follows:

$$\mathbb{E}_X[X] = \frac{\alpha}{\alpha + \beta} \quad \mathbb{E}[(X - \mu_X)^2] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}.$$

Applications The Beta distributions is often used in Bayesian statistics, since it is the “natural” prior to the binomial distributions. See http://varianceexplained.org/statistics/beta_distribution_and_baseball/ and <https://stats.stackexchange.com/a/47782>

Relations to other distributions

- If we set $\alpha = \beta$ and both go to infinity, the beta distribution becomes the normal: $\text{Beta}(\alpha, \beta) = \mathcal{N}(\dots)$
- $\text{Beta}(\alpha = 1, \beta = 1) = \mathcal{U}(0, 1)$

[The beta function]

https://en.wikipedia.org/wiki/Beta_function

[The beta probability distribution]

https://en.wikipedia.org/wiki/Beta_distribution

Cauchy (optional)

The probability density function for the random variable $X \sim \text{Cauchy}(x_0, \gamma)$ is

$$f_X(x) = \frac{1}{\pi\gamma} \frac{1}{1 + \left(\frac{x-x_0}{\gamma}\right)^2},$$

where $x_0 \in \mathbb{R}$ and $\gamma \in \mathbb{R}_+$.

The parameter x_0 is called the shape parameter, while γ is the inverse scale parameter.

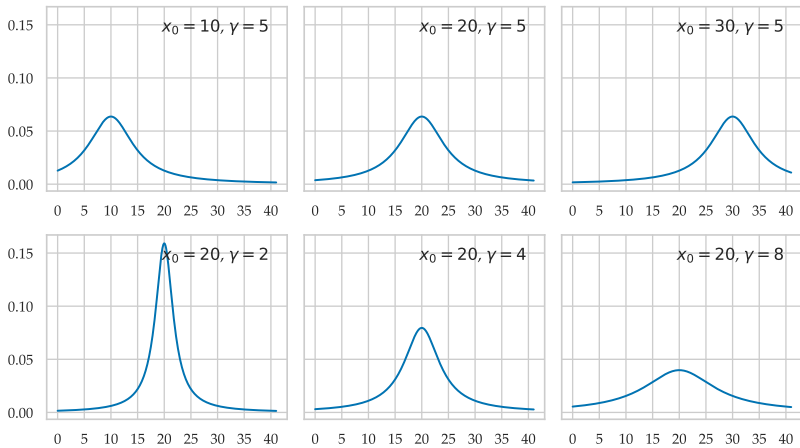


Figure 2.85: Plot of the Cauchy distribution for different choices of the parameters x_0 and γ .

NO mean no var!

Exercises:

- Show $\mathbb{E}_{XY}[X/Y]$ doesn't exist.

Relations to other distributions

- If X and Y that are two normally distributed random variables with mean 0 and standard deviation 1, $X \sim \mathcal{N}(0, 1)$ and $Y \sim \mathcal{N}(0, 1)$. Then the random variable that corresponds to the ratio X/Y has the Cauchy distribution: $\frac{X}{Y} \sim \text{Cauchy}(0, 1)$.
- Special case of Student's t -distribution when $\nu = 1$.

[The Cauchy distribution]

https://en.wikipedia.org/wiki/Cauchy_distribution

2.6.3 Modelling real-world data

Now that we've learned about continuous probability distributions, let's look back at the two datasets we introduce in the book's introduction, to see how probabilistic modelling skills we developed can help you better understand datasets.

Dataset 2: Electricity prices data

Recall Bob's dataset of electricity prices available for electric car charging in the East and West parts of his city. See page ??.

The electricity price at each station depends on numerous variables (supply cost, rent, location, etc.) so one could make an argument that if the price is governed by the sum of these contributions, then it will be Gaussian, or normally distributed, and each price observation is independent.

Bob can use the average price in the East is $\bar{x}_E = 6.156$ ¢/kWh and the sample variance s_{X_W} , and the average in the West is $\bar{x}_W = 9.156$ ¢/kWh and the sample variance s_{X_W} , as the basis of constructing probability models X_E and X_W , that describe prices in the East and West.

One could also argue that prices are not normally distributed or independent. CONTINUE

It seems we need some tools to check the *normality assumption* ... We'll talk more about comparing data observations for "model fit" to a theoretical distribution later in Section 2.7.

TODO: mention Student t-distribution also useful for estimates...

Dataset 3: Student grades data

Recall the students' grades dataset that Charlotte collected in order to study the effectiveness of a new teaching method. See page ??.

We can model the effort and score variables as a normally distributed, with parameters μ and σ that describe the shape of the data.

We can then compare the difference between teaching methods, by comparing the probability distributions.

TODO: mention Student t-distribution also useful for estimates...

2.6.4 Discussion

Location, scale, and standardization Continuous probability distributions are often described in terms of a location (where is the peak) and a scale parameter (how spread out is the density).

The canonical example of this is the normal distribution $N \sim \mathcal{N}(\mu, \sigma)$, whose location parameter is the mean μ and whose scale parameter is the standard deviation σ .

The standardization transformation for a random variable X is

$$Z = \frac{X - \mu_X}{\sigma_X}$$

We can similarly do this transformation on `x` `x_std = (xsample - xsample.mean()) / xsample.std ...` TODO: finish this mention (or cut) this is bonus so “standardize” is not first time mention in Section 2.8

Summary of relations between distributions

TODO FIGURE FULL GRAPH (including discrete and continuous)

TODO: insert simplified concept map from <http://www.stat.rice.edu/~dobelman/courses/texts/leemis.distributions.2008amstat.pdf#page=3> or <https://pdfs.semanticscholar.org/c0db/71a4101347404d698f68fbed54ddb88b1500.pdf#page=2>

Normal approximation to the binomial distribution If X is a binomial random variable with parameters n and p , then

$$Z = \frac{X - np}{\sqrt{np(1-p)}}$$

is approximately equal to the standard normal. To approximate a binomial probability with a normal distribution, a continuity correction is given by

$$\Pr(X = k) = \Pr(k - 0.5 \leq X \leq k + 0.5) \approx \Pr\left(\frac{k - 0.5 - np}{\sqrt{np(1-p)}} \leq Z \leq \frac{k + 0.5 - np}{\sqrt{np(1-p)}}\right)$$

$$\Pr(X \leq k) = \Pr(X \leq k + 0.5) \approx \Pr\left(Z \leq \frac{k + 0.5 - np}{\sqrt{np(1-p)}}\right)$$

$$\Pr(X \geq k) = \Pr(X \leq k - 0.5) \approx \Pr\left(Z \geq \frac{k - 0.5 - np}{\sqrt{np(1-p)}}\right)$$

This approximation is good for $np > 5$ and $n(1-p) > 5$.

Normal approximation to the Poisson distribution If X is a Poisson random variable with $E[X] = \lambda$ and $V[X] = \lambda$, then

$$Z = \frac{X - \lambda}{\sqrt{\lambda}}$$

is approximately a standard normal random variable. The same continuity correction used for the binomial distribution can also be applied. The approximation is good for $\lambda > 5$.

TODO: give other examples of phenomena to show normality emerge for large n

<https://www.efavdb.com/normal-distributions>

Reminder of computer model methods

The inventory of continuous distributions presented above contains code examples for creating a random variable object `rvX` from any of the following families of probability distributions: `uniform`, `expon`, `norm`, `t`, `f`, `chi2`, `gamma` and `beta`, all defined in `scipy.stats`.

We showed only basic calculations in the code examples, but you should keep in mind that there are a lot of other methods available on random variable objects created from one of the families in `scipy.stats`. To see a complete list of all the methods available, look back to Table 2.2 on page 141. You'll need to use some of these methods to complete the exercises.

Recall how we compute probabilities numerically. The probability $\Pr(\{a \leq X \leq b\})$ is defined as the integral $\int_a^b f_X(x)dx$. You can compute the value of this integral numerically by calling the function `quad(rvX.pmf, a, b)`, after importing `quad` from the module `scipy.integrate`.

2.6.5 Exercises

E2.37 Calculate the value of $\Gamma(z) \stackrel{\text{def}}{=} \int_0^\infty t^{z-1} e^{-t} dt$, when $z = 1$ by evaluating the integral.

Hint:

E2.38 Show that $\Gamma(z) = (z-1)\Gamma(z-1)$, by applying the integration-by-parts procedure to the integral $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$.

Hint:

E2.39 Calculate the mean of the uniform distribution $\mathcal{U}(\alpha, \beta)$.

E2.40 The probability density function of the random variable $X \sim \text{Expon}(\lambda)$ is given by $f_X(x) = \lambda e^{-\lambda x}$. Use integration to obtain the formula for cumulative distribution function $F_X(b) = \Pr(\{X \leq b\})$.

E2.41 Use integration to compute the mean of the exponential distribution $f_X(x) = \lambda e^{-\lambda x}$.

Hint: Use integration by parts.

E2.42 Use integration to compute the variance of the exponential distribution $f_X(x) = \lambda e^{-\lambda x}$.

Hint: Use integration by parts.

Links

[Read more about the relations between probability distributions]

https://wikipedia.org/wiki/Relationships_among_probability_distributions

[Complete list of the continuous distributions available in SciPy]

<https://docs.scipy.org/doc/scipy/tutorial/stats/continuous.html>

[More info about the gamma and beta functions]

<https://docs.scipy.org/doc/scipy/reference/special.html>

2.7 Random variable generation

In this section, we'll learn how to use computers to generate observations from random variables. Using computer simulations for random variables is a very useful tool for learning and visualizations, which we'll use throughout the rest of the book.

Suppose you want to generate a random observation x from the random variable X , described by the probability distribution f_X . One approach would be to create a random variable object `rvX` based on one of the model families defined in `scipy.stats`, then call the method `rvX.rvs()` to generate a random observation. But how does the random generation process work under the hood? And how do you know if a sequence of observations (x_1, x_2, \dots, x_n) really comes from the distribution f_X ?

2.7.1 Definitions

Let's start by introducing the concepts we'll use in this section:

- X : a random variable with probability distribution f_X
- `gen_x`: a function that *generates* random observations x from the random variable X .
- $\mathbf{x} = (x_1, x_2, \dots, x_n)$: a *sample* of n observations generated by `gen_x`. If the generator function `gen_x` is working correctly, the distribution of the observations x_i will correspond to the distribution f_X of the random variable X .
- $f_{\mathbf{x}}$: the *empirical probability mass function* (epmf) of the sample \mathbf{x} . The empirical distribution allows us to model the data observations (x_1, x_2, \dots, x_n) as a probability distribution.
- $F_{\mathbf{x}}$: the *empirical cumulative distribution function* (eCDF) of the sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$. The function $F_{\mathbf{x}}$ is the integral of $f_{\mathbf{x}}$.

2.7.2 Why simulate?

Using computers to simulate observations from random variables is an essential skill I want you to develop, since it will be very useful for understanding statistics procedures and verifying math equations “experimentally.”

In the statistics chapter (Chapter ??) we'll study all kinds of statistical procedures for modelling real-world data using probability distributions. We'll have to learn complicated, multi-step procedures that involve lots of math equations and formulas. Some of the math formulas will look quite intimidating! It's easy to get lost in all the math and get discouraged.

Simulations to the rescue! Running computer simulations gives us a hands-on alternative approach to understanding statistical procedures, and verifying the validity of math equations. For example, suppose that a statistician has come up with a math formula for the quantity $\mathbb{E}_X[g]$, which is the expected value of some complicated function $g : \mathcal{X} \rightarrow \mathbb{R}$ under the randomness of the random variable X described by the probability distribution f_X . The statistician computed some complicated math integral $\mathbb{E}_X[g] = \int_{x \in \mathcal{X}} g(x) \cdot f_X(x) dx$, and is offering you the result of the integral as a pre-packaged formula you can use whenever you need to compute $\mathbb{E}_X[g]$.

In the old days before computers were widely available, you'd have to take it on faith that the statistician did the math correctly and the formula works as expected. In modern times when computers are everywhere (often in your pocket!), you can run a computer simulation to verify the statistician's claim.

Suppose you have access to a generator function `gen_x()` that generates random observations from the probability distribution f_X . You can call the generator function `gen_x()` multiple times to obtain a sequence of observations $(x_1, x_2, x_3, \dots, x_n)$. Next you can compute the function g on each of these observations to obtain a list of g -values $[g(x_1), g(x_2), g(x_3), \dots, g(x_n)]$. You can then obtain an estimate of the expected value $\mathbb{E}_X[g]$ by computing the average of the list of g -values.

In essence, running a simulation based on the observations $(x_1, x_2, x_3, \dots, x_n)$ gives us an alternative way to compute the same quantity $\mathbb{E}_X[g]$ using summation instead of integration. We can describe what is going on using the following equation:

$$\mathbb{E}_X[g] = \int_{x \in \mathcal{X}} g(x) \cdot f_X(x) dx \approx \frac{1}{n} \sum_{i=1}^n g(x_i).$$

The summation formula on the right side of this equation is an estimate of the expected value $\mathbb{E}_X[g]$, computed from a finite sample of observations from f_X . As n becomes larger and larger, the approximation becomes more and more accurate. In the limit $n \rightarrow \infty$ (read n goes to infinity), the \approx sign becomes an equality.

Let's now look at the code for running a simulation involving $n = 100$ random observations x_i from X , and computing the expected value $\mathbb{E}_X[g]$ using the formula $\frac{1}{n} \sum_{i=1}^n g(x_i)$.

```
>>> n = 100
>>> gvalues = []
>>> for i in range(0,n):
>>>     xi = gen_x()
>>>     gi = g(xi)
>>>     gvalues.append(gi)
>>> sum(gvalues) / n
```

code
2.7.1

The code maps directly to the math summation formula. The main structure is the `for`-loop which repeats the simulation steps n times. In each iteration of the `for` loop, we call the generator function `gen_x` to obtain the random observation $x_i = x_i$, compute the value $g_i = g(x_i)$, then add the value g_i to the end of the list `gvalues`. On the last line, we finally compute the average of the list `gvalues`, which corresponds to the desired result $\frac{1}{n} \sum_{i=1}^n g(x_i) \approx \mathbb{E}_X[g]$.

* * *

Essentially, the reason simulations are so important, is that all the probability modelling and estimation you might want to do with the probability model f_X , you can verify running a simulation on a large number of observations $(x_1, x_2, x_3, \dots, x_n)$ in a numerical simulation.

Using a few lines of Python code, you can create a simulation of any probability scenario, and use the results of your simulation to visualize the data distributions involved and calculate quantities of interest. Think about all the different probability distributions we learned about in this chapter, and the complicated math formulas associated with them. If you know how to write a `for` loop in Python, then you can generate lots of random observations from the random variable, then plot a histogram to see what is going on. In other words, simulations allow you to see every distribution and verify every equation.

All this to say that simulations are good. And what do you need to run simulations involving random variables? You need some way to generate observations from random variables, which is the topic we'll learn about next.

2.7.3 Random variable generation using a computer

Most programming languages provide some way to generate random numbers from the standard uniform distribution $\mathcal{U}(0, 1)$. The ability to generate random observations from the uniform distribution is an essential building block that allows us to generate random observations from any other distribution. In Python, you can import the `random` module then generate random numbers between 0 and 1 by calling the function `random.random()` as shown below:

```
>>> import random
>>> random.random()
0.8371523646930408
```

code
2.7.2

Every time you call the function `random.random()`, you'll see a different random number in the interval $[0, 1]$. In other words,

calling the function `random.random()` is equivalent to generating an observation from the uniform random variable $U \sim \mathcal{U}(0, 1)$.

In this section, we'll describe how to generate random observations from any probability distribution based on the standard uniform randomness provided by `random.random()`. The practical importance of knowing how to generate random variables is of marginal utility (since we already have the method `rvX.rvs()`), but we present this topic because of the associated “math tools” that we'll need to develop to “verify” if the random variable generation procedures we use are working correctly.

Discrete random variable generation

Starting from the standard uniform random variable $U \sim \mathcal{U}(0, 1)$, we can generate any discrete random variable by “slicing” the interval $[0, 1]$ appropriately.

Example 1: observations from the Bernoulli distribution Suppose we want to generate outcomes from the distribution $\text{Bernoulli}(p)$, which is equivalent to a coin toss of a biased coin that has probability of coming out heads p . We'll start by drawing a uniform random number u from `random.random()`, then return 1 (heads) if the random number r is smaller than p , or else return 0 (tails).

```
>>> def gen_b(p=0.5):
    u = random.random()
    if u < p:
        return 1
    else:
        return 0
```

code
2.7.3

The function `gen_b` expects the argument p to be specified, which is the probability of “heads” of the Bernoulli trial. If p is not specified, the default value $p=0.5$ is used (equivalent to a fair coin).

Since the random number u is uniformly distributed between 0 and 1, the observed outcome will be 1 with probability p , and 0 with probability $(1 - p)$.

To generate a random observation from the distribution $\text{Bernoulli}(p = 0.3)$, we simply call the generator function, specifying the value of the parameter p :

```
>>> gen_b(p=0.3)
0
```

code
2.7.4

The code below generates a list of $n=100$ observations from the distribution $\text{Bernoulli}(p = 0.3)$, and computes the proportion of the outcome 1 (heads) in the list.

code
2.7.5

```
>>> n = 1000
>>> bsample = [gen_b(p=0.3) for i in range(0,n)]
>>> bsample.count(1) / n
0.309
```

Note the proportion of heads observed after 1000 coin tosses is approximately equal to the parameter $p = 0.3$, which we specified for the Bernoulli distribution, so it seems the random generation function `gen_b` is working correctly.

We can use this approach to generate random observations from any discrete random variable Y . We just need to know the values of the cumulative distribution function F_Y , which tell us the appropriate places to “slice” the sample space $[0, 1]$ of the standard uniform random variable.

Continuous random variable generation (optional)

Suppose we want to generate observations from a continuous random variable X , and we know its inverse cumulative distribution function F_X^{-1} . We can simulate observations from X by transforming the uniform random variable $U \sim \mathcal{U}(0, 1)$ through the inverse CDF function. We can write this as:

$$X = F_X^{-1}(U).$$

In other words, generating observations from the random variable X can be done by starting from observations from the uniform distribution, and “passing them through” the inverse cumulative distribution function F_X^{-1} . This procedure is known as *inverse transform sampling* or *Smirnov transform*, but we’ll refer to it as the “inverse-CDF trick” in this book.

The inverse-CDF trick corresponds to the following two-step procedure:

1. Generate an observation u from the standard uniform distribution $U \sim \mathcal{U}(0, 1)$
2. Compute the value $x = F_X^{-1}(u)$ (which is equivalent to solving the equation $F_X(x) = u$).

The result x will be a random observation from the random variable X . Let’s look at some examples of the inverse-CDF trick in practice.

Example 2: shifted uniform distribution Suppose we want to generate observations from the random variable $V \sim \mathcal{U}(\alpha = 100, \beta = 120)$, which is uniformly distributed between 100 and 120.

TODO: add steps to obtain F_V from f_V and invert to obtain $F_V^{-1}(q)$

```
>>> def gen_v():
    u = random.random()
    v = 100 + 20*u
    return v
>>> gen_v()
110.89
>>> n = 100 # sample size
>>> vsample = [gen_v() for i in range(0,n)]
>>> sns.histplot(vsample, stat="density")
```

code
2.7.6

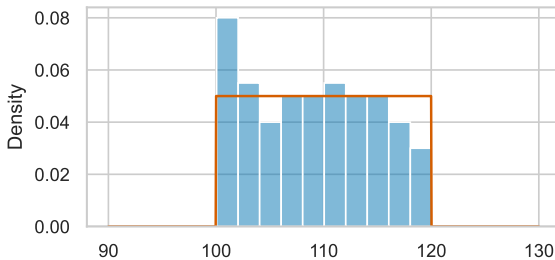


Figure 2.86: Histogram of 1000 observations from the shifted uniform distribution `vsample`, which we generated using the inverse-CDF trick. The superimposed line shows the probability density function of the random variable $V \sim \mathcal{U}(\alpha = 100, \beta = 120)$.

Figure 2.86 shows a histogram of the values in the list `vsample`. Using visual inspection, we see all the numbers we generated fall in the right range $[100, 120]$, and look roughly uniform (except for one of the histogram bin).

Example 3: exponential distribution Suppose we want to generate random observations from the exponential random variable $E \sim \text{Expon}(\lambda = 0.2)$. The random variable E is described by

the probability density function $f_E(x) = \lambda e^{-\lambda x}$ and its cumulative distribution function is $F_E(b) = 1 - e^{-\lambda b}$ (see Exercise E2.40).

To apply the “inverse-CDF trick,” we need to know the inverse cumulative distribution F_E^{-1} , which requires some math calculations, which involve solving for b in the equation $F_E(b) = q$. You’ll be asked to do these calculations in E2.43. The inverse cumulative distribution function of the random variable E is $F_E^{-1}(q) = -\frac{\ln(1-q)}{\lambda}$.

We can now use the formula $F_E^{-1}(q) = -\frac{\ln(1-q)}{\lambda}$ to transform observations from the uniform distribution to obtain observations from the exponential distribution. We’ll generate $n = 1000$ such observations, and plot a histogram of them.

```
>>> def gen_e(lam):
    u = random.random()
    e = -1 * np.log(1-u) / lam
    return e
code
2.7.7

>>> n = 100 # sample size
>>> esample = [gen_e(lam=0.2) for i in range(0,n)]
>>> sns.histplot(esample, stat="density")
```

Figure 2.87 shows the histogram of the observations in `esample`. For comparison, we have also plotted the probability density function of the random variable $E \sim \text{Expon}(\lambda = 0.2)$, which is the distribution we are trying to simulate. As you can see, the distribution of the data in `esample` is similar to the desired model f_E , so it seems the random variable generator function `gen_e` is working correctly.

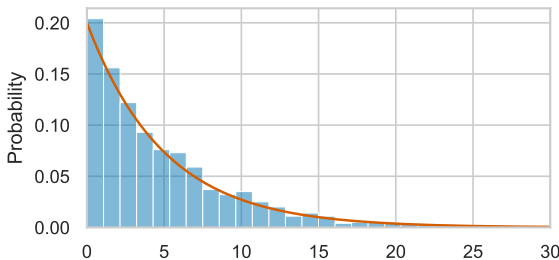


Figure 2.87: Histogram of 100 observations from the random variable $E \sim \text{Expon}(\lambda = 0.2)$ obtained using the inverse-CDF trick. The superimposed line shows the probability density function $f_E(x) = \lambda e^{-\lambda x}$.

* * *

You can use the inverse-CDF trick to generate random observations from any continuous random variable X if you know its inverse cumulative distribution function F_X^{-1} .

How can we know if the random variable generation procedures we used in the above examples are accurate? In the above examples we did a “visual comparison” of the histograms of the observations in `vsample` and `esample`, and we saw they roughly correspond to the probability density function f_V and f_E , but is there a more systematic approach we could use to check if the generation procedure is working correctly?

The ability to check whether a given sample of observations (data) comes from a particular distribution (model) is an important task in statistics, which we refer to as *data-model fit* or more generally *goodness of fit*. We’ll continue the discussion of *data-model fit* checks in a few pages, but first we need to introduce some math tools for describing data distributions using the language of probability distributions.

2.7.4 Empirical distribution of a data sample

The *empirical distribution* allows us to describe data from a sample using the tools of probability theory. The term “empirical” means the same thing as “experimental” and refers to real data we have observed (or generated in a simulation). We need to use the modifier “empirical” in front of “distribution” because by default the meaning of “distribution” in probability and statistics is a “theoretical distribution” or “model,” which is described using math equations.

Given a sample of n values $\mathbf{x} = (x_1, x_2, \dots, x_n)$, the *empirical probability mass function* (epmf) of \mathbf{x} is defined as:

$$f_{\mathbf{x}}(x) \stackrel{\text{def}}{=} \begin{cases} \frac{1}{n} & \text{if } x = x_i, \\ 0 & \text{otherwise.} \end{cases}$$

In words, the empirical distribution $f_{\mathbf{x}}$ places probability mass $\frac{1}{n}$ on each of the values x_1, x_2, \dots, x_n , and zero mass everywhere else.

Recall that every random variable X can be described either in terms of its probability distribution f_X , or, equivalently, in terms of its cumulative distribution function $F_X(b) \stackrel{\text{def}}{=} \Pr(\{x \leq b\})$. The *empirical cumulative distribution function* (eCDF) obtained based on the sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is defined as follows:

$$F_{\mathbf{x}}(b) = \frac{\text{count}(x_i \leq b)}{n}.$$

The empirical cumulative distribution $F_{\mathbf{x}}$ computes the proportion of observations that are less than or equal to the upper limit b .

Let’s define a Python function `ecdf(data, b)` that computes the values of the empirical cumulative distribution function $F_{\text{data}}(b)$.

```
>>> def ecdf(data, b):
    sdata = np.sort(data)
    count = sum(sdata <= b)      # num. of obs. <= b
    return count / len(data)    # proportion of total
```

In words, we see that calculating $F_x(b)$ is equivalent to counting the number of observations x_i that are less than or equal to b , and dividing by the total n .

Let's use try the function

```
>>> ecdf(vsample, 110)
0.48
```

code
2.7.9

This is expected, since the value 110 falls roughly in the middle of the sample space of the random variable V .

Figure 2.88 shows the graph of the empirical cumulative distribution F_{vsample} and the probability mass function f_V .

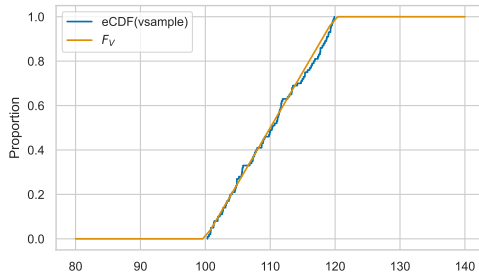


Figure 2.88: TODO

Figure 2.89 shows the graph of the empirical cumulative distribution F_{esample} and the probability mass function f_E .

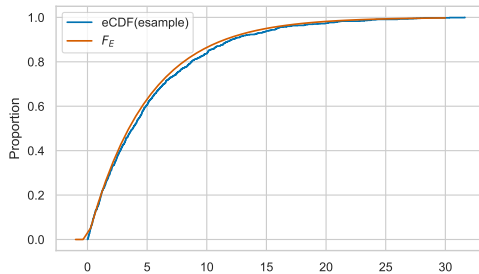


Figure 2.89: TODO

TODO: narrate figs

Applications

The empirical distribution of the data sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is a useful mathematical “adapter” that allows us to describe all kinds of data using the language of probability theory (probability distributions). The sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ could be produced from any one of the following processes:

- Random draws from a generative process or simulation like the Python functions like `gen_b` and `gen_e` we defined above.
- Random draws from a real-world process (real-world measurements).
- A sample from a population (real-world observations from a sample)

Note this list contains many different kinds of variability and randomness: randomness obtained from simulations, and measurements obtained from entire population or samples from populations. By representing a real-world data source as an empirical distribution $f_{\mathbf{x}}$, we can then use all the tools we learned in this chapter to make calculations. We’ll see empirical distributions again in Chapter ??.

Another application of the empirical distribution is to generate *bootstrap* samples, which is the process of sampling from the empirical distribution. We’ll talk more about the bootstrap at the end of this section (see page 215).

For now, let’s focus on the original question related to random variable generation, how can we know if the data we generated using the inverse-CDF trick is working correctly? Are the random numbers we generated (x_1, x_2, \dots, x_n) really coming from the distribution f_X ?

2.7.5 Measuring data–model fit

Suppose you have a sample of observations $\mathbf{x} = (x_1, x_2, \dots, x_n)$, and you want to check if these observations come from the theoretical distribution f_X . We call this a *data–model fit* question, since we want to check if the data \mathbf{x} fits the model f_X . We can rephrase this question in terms of the empirical distribution $f_{\mathbf{x}}$ obtained from the sample \mathbf{x} , by asking “how similar is the distribution $f_{\mathbf{x}}$ to the theoretical distribution f_X ,” or equivalently, “how similar is the empirical CDF $F_{\mathbf{x}}$ to the CDF of the theoretical distribution F_X .”

Recall the lists of observations `vsample` and `esample` that we generated using the inverse-CDF trick. In the next few pages, we’ll learn about various “diagnostic checks” to assess whether `vsample`

comes from the model $V \sim \mathcal{U}(100, 120)$, and we'll also check if `esample` comes from the model $E \sim \text{Expon}(\lambda = 0.2)$.

In order to have a greater variety of distributions for use in comparisons, let's generate an additional sample of 100 observations from the normal distribution $N \sim \mathcal{N}(\mu = 1000, \sigma = 100)$.

```
>>> from scipy.stats import norm
>>> rvN = norm(1000, 100)
>>> nsample = rvN.rvs(100)
```

code
2.7.10

We know the data in `nsample` comes from the distribution f_N , so we should expect all the data-model fit diagnostics checks between `nsample` and f_N to pass.

Visual comparison between data and model distributions

Earlier in this section, we used a visual comparison between the histogram of the samples `vsample` and `esample` and the theoretical probability density function f_V and f_E . See Figure 2.86 and Figure 2.87. We also computed the empirical cumulative distribution function from the generated samples, and compared them to the cumulative distribution of the desired models. Look back to Figure 2.88 and Figure 2.89.

These types of visual comparisons can help detect major differences between data and model, but there are more precise qualitative and quantitative techniques we can develop.

Quantile-quantile plots

The quantile-quantile plot (Q-Q plot) is a more advanced version of a visual comparison, that plots the quantiles of the empirical distribution against the quantiles of a theoretical distribution. If the sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ comes from the distribution f_X , then all the points in the Q-Q plot should fall on a diagonal line.

Examples of good fit Let's start with the normal data `nsample`, which we know comes from the distribution $\mathcal{N}(\mu = 1000, \sigma = 100)$. Figure 2.90 shows the Q-Q plot that compares the empirical distribution obtained from `nsample` and the theoretical model $\mathcal{N}(1000, 100)$.

```
>>> from statsmodels.graphics.api import qqplot
>>> from scipy.stats import norm
>>> qqplot(nsample, dist=norm(1000, 100), line='q')
The result is shown in Figure 2.90.
```

code
2.7.11

We often don't need to check for exact fit with a specific distribution, but just want to check if the data comes from the same family of distributions. Are the observations in `nsample` normally

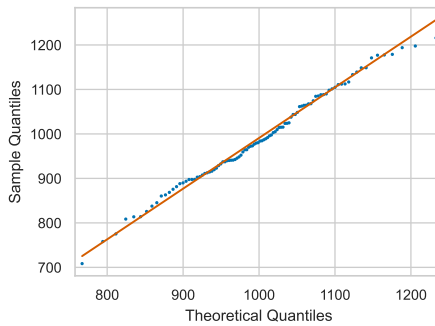


Figure 2.90: Q-Q plot showing the quantiles of the normally distributed data sample `nsample` against the quantiles of the normal distribution $\mathcal{N}(1000, 100)$. We see all the points lie very close to the diagonal line, so the two distributions are very similar.

distributed? To do this check (sometimes called a *normality check*), we generate a Q-Q plot with respect to the standard normal, as shown in Figure 2.91.

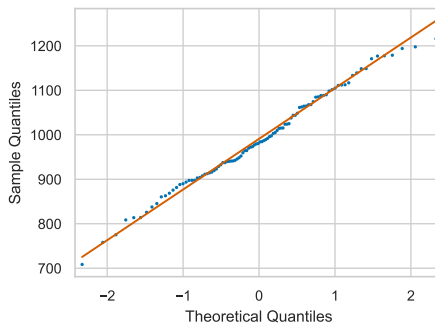


Figure 2.91: Q-Q plot of the normally distributed data sample `nsample` against the quantiles of the standard normal distribution $\mathcal{N}(0, 1)$.

The code to generate Figure 2.91 is very similar to the code we used to generate Figure 2.90, but we pass in the model `norm(0, 1)` as the `dist` argument:

```
>>> qqplot(nsample, dist=norm(0,1), line='q')
See Figure 2.91.
```

code
2.7.12

Note the Q-Q plot in Figure 2.91 is identical to the plot in Figure 2.90 except for the scale on the horizontal axis.

Examples of bad fit Let's now see what happens when the data distribution doesn't match the theoretical model. We know the datasets `vsample` and `esample` are *not* normally distributed, so we

should be able to detect that by looking at the Q-Q plots against a normal distribution.

```
>>> vsample = np.array(vsample)
>>> qqplot(vsample, dist=norm(0,1), line="q")
See Figure 2.92 (a).
```

code
2.7.13

```
>>> esample = np.array(esample)
>>> qqplot(esample, dist=norm(0,1), line="q")
See Figure 2.92 (b).
```

The lines where we convert `vsample` and `esample` to NumPy arrays of the same name are necessary because the function `qqplot` assumes some methods are present on the data inputs.

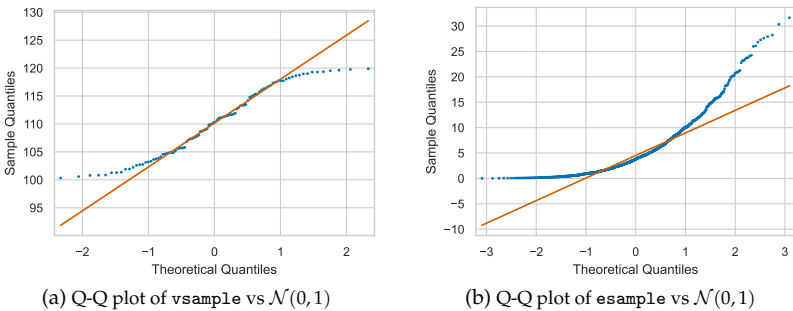


Figure 2.92: Q-Q plots showing examples where the quantiles of a data distribution do not fit quantiles of the theoretical model. Subfigure (a) shows the quantiles of the uniformly distributed data `vsample` against the quantiles of the standard normal $\mathcal{N}(0,1)$. Subfigure (b) shows the quantiles of observations in the list `esample` against the standard normal.

Figure 2.92 (a) shows the Q-Q plots of the uniform observations `vsample` plotted against the standard normal. The distribution of the data in `vsample` is bounded between 100 and 120, while the normal distribution has long tails, so we see major discrepancies in the Q-Q plot in the tails. In Figure 2.92 (b) we see the Q-Q plot of the exponential observations `esample` against the quantiles of the standard normal. Again, we can see that very few points are close to the diagonal. We can conclude from the Q-Q plots that the observations `vsample` and `esample` do not match the quantiles of the normal model.

Having seen examples of Q-Q plots that show “good fit” and “bad fit” between data and model, we can now look at the goodness of fit between the data in `vsample` we generated using the inverse-CDF trick, and the theoretical model $V \sim \mathcal{U}(100, 120)$, which is shown in Figure 2.93.

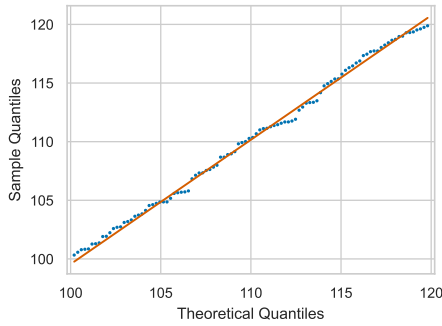


Figure 2.93: Q-Q plot of the random observations we generated `vsample` using the inverse-CDF trick and comparison to quantiles the random variable $\text{rvV} \sim \mathcal{U}(100, 120)$.

Figure 2.94 shows the comparison between the exponential data `esample` against the exponential model $\text{Expon}(\lambda = 0.2)$.

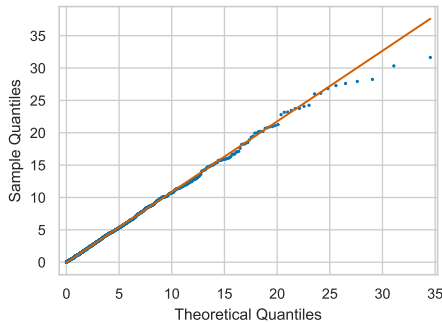


Figure 2.94: Q-Q plot of the data from `esample` and the random variable $\text{rvE} \sim \text{Expon}(\lambda = 0.2)$.

Except for a few points, most of the data points in both plots fall very close to the diagonal, so we conclude there is a good fit between the data and the theoretical model. The random variable generation process is working correctly.

Comparing moments

Another way to measure how well the data sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ fits the probability model f_X is to check if the data sample and the probability distribution have the same *moments*.

Recall that the moments of a distribution are the expectation of $(X - \mu_X)$ raised to different exponent. The m^{th} moment of the distribution f_X around its mean μ_X is defined as $\mathbb{E}_X[(X - \mu_X)^m]$. The variance of X is computed as the second moment $\sigma_X^2 = \mathbb{E}_X[(X -$

$\mu_X)^2]$. The skewness of X is related to the third moment of the distribution $\mathbb{E}_X[(X - \mu_X)^3]$, and the kurtosis of X is computed based on the fourth moment $\mathbb{E}_X[(X - \mu_X)^4]$.

The moments of a data sample are computed as summations involving $(x_i - \bar{x})$ raised to different exponent. The m^{th} moment of a data sample is computed as $\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^m$, and we compute the variance (second moment), skewness (related to third moment), and kurtosis (related to fourth moment) for any data sample.

If the data sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ comes from the distribution f_X , then all moment should match. In particular, we would expect the sample mean to be equal to the distribution mean, and the sample variance to be equal to the distribution variance. We can therefore measure model-data fit by answering the question:

$$\text{Is } \mathbf{Mean}(\mathbf{x}) = \mu_X \text{ and } \mathbf{Var}(\mathbf{x}) = \sigma_X^2 \text{ ?}$$

The skewness and kurtosis computed from the data sample should also match the skewness and kurtosis of the distribution. We can do these four comparisons to assess the goodness of fit between the data \mathbf{x} and the model f_X .

In order to easily be able to calculate the moments of the data samples, we'll convert `vsample` and `esample` into Pandas series objects, which have all the necessary methods.

```
>>> vseries = pd.Series(vsample)
>>> eseries = pd.Series(esample)
```

code
2.7.14

Moments of uniform distribution Let's compare the similarity between moments of data sample `vsample` and the theoretical distribution $V \sim \mathcal{U}(100, 120)$, represented by the computer model `rvV = uniform(100, 20)`.

Does the mean of the `vseries` match the mean of the `rvV`? We can find out by calling the `mean` method on the object `vseries` and `rvV`.

```
>>> vseries.mean(), rvV.mean()
(110.212, 110.0)
```

code
2.7.15

We see the mean of the data we generated closely matches the mean of the theoretical distribution f_V , which is a good sign.

Does the variance of `vseries` match the variance of `rvV`?

```
>>> vseries.var(), rvV.var()
(35.95, 33.3)
```

code
2.7.16

We can see that both the mean and the variance of the data we generated closely match the mean and the variance of the theoretical f_V . We continue the comparisons, by calculating the skew and the kurtosis:

```
>>> vseries.skew(), rvV.stats("s")
(0.0397, 0.0)
>>> vseries.kurt(), rvV.stats("k")
(-1.258, -1.2)
```

Again we see a close match between the properties of data observations in `vseries` and the theoretical model `rvV`, so we can conclude that the random generation procedure we used to generate `vseries` is working as expected.

Moments of exponential distribution We'll now compare the similarity between the moments of the data sample `esample` and the moments of the theoretical distribution $E \sim \text{Expon}(\lambda = 0.2)$, represented by the computer model `rvE = expon(0, 1/0.2)`.

The code below computes the mean, the variance, the skewness, and the kurtosis, of both data and model:

```
>>> eseries.mean(), rvE.mean()
(5.234, 5.0)
>>> eseries.var(), rvE.var()
(25.157, 25.0)
>>> eseries.skew(), rvE.stats("s")
(1.931, 2.0)
>>> eseries.kurt(), rvE.stats("k")
(5.458, 6.0)
```

code
2.7.18

We can see there is a clear agreement between the moments of the data `esample` and the theoretical distribution `rvE`, so we can be reassured that the observations in `esample` really come from the distribution we were trying to simulate.

Example of bad fit between higher moments Let's now look at what happens if we compare the moments of the exponential data `eseries` to the moments of a wrong theoretical model. Define the random variable N_E that is normally distributed with mean and standard deviation matching the theoretical model E :

$$N_E \sim \mathcal{N}(\mu = 5, \sigma = 5).$$

In words, N_E is the best normal approximation to the exponential distribution $E \sim \text{Expon}(\lambda = 0.2)$.

Let's also define the computer model `rvNE` that corresponds to the random variable N_E .

```
>>> rvNE = norm(5,5)
>>> rvNE.mean(), rvNE.var()
(5.0, 25.0)
```

code
2.7.19

If we compare the mean and the variance of the sample `esample` to the mean and variance of the model N_E we'll see there is a close

agreement. This is because we intentionally chose the parameters of the normal approximation N_E so they match those of the exponential distribution: $\mu_{N_E} = \mu_E = \frac{1}{\lambda} = 5$ and $\sigma_{N_E}^2 = \sigma_E^2 = \frac{1}{\lambda^2} = 25$.

Let's now compare the skewness of `eseries` and the skewness of the normal random variable N_E :

```
>>> eseries.skew(), rvNE.stats("s")
(1.931, 0.0)
```

code
2.7.20

We observe a big disagreement between the third moments. The normal distribution N_E is symmetric, so it has zero skewness, but the data in `eseries` has positive skewness (a long tail that extends to the right), and we're able to detect this data-model mismatch thanks to the skewness comparison.

We can also compare the kurtosis of the data in `esample` and the kurtosis of the distribution N_E .

```
>>> eseries.kurt(), rvNE.stats("k")
(5.458, 0.0)
```

code
2.7.21

Again we observe a big disagreement. The normal distribution has zero kurtosis, but the exponential distribution has positive kurtosis. Informally, we could say the data in `esample` has heavier tails as compared to the tails of a normal distribution.

Due to the differences observed in the third and fourth moments, we conclude that the data in `esample` (`eseries`) does not come from the distribution $\mathcal{N}(\mu = 5, \sigma = 5)$.

Later on in Chapter ?? we'll learn about the "method of moments," which is a way to find the "best fit" model to a given dataset. The method of moments is based the idea of choosing the parameters for the model so the mean and variance will match the mean and variance of the data.

Kolmogorov–Smirnov distance

We can measure the "difference" between the sample of observations $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and the probability distribution f_X using the Kolmogorov–Smirnov distance. The Kolmogorov–Smirnov distance, denoted D_{KS} , is defined as the maximum difference between the empirical cumulative distribution function $F_{\mathbf{x}}$ obtained from the sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$, and the CDF of the theoretical distribution F_X :

$$D_{KS} = \max_b |F_{\mathbf{x}}(b) - F_X(b)|.$$

If the quantity D_{KS} is large, this means there is some value of b for which the empirical cumulative distribution function $F_{\mathbf{x}}$ differs

significantly from the theoretical cumulative distribution function F_X .

Figure 2.95 illustrates the maximum distance between the empirical cumulative distribution obtained from some dataset x and the cumulative distribution function F_X of the theoretical model.

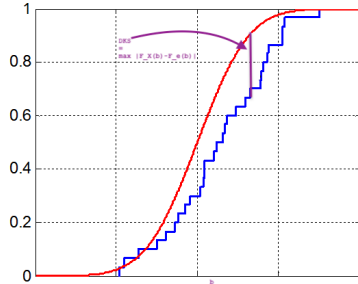


Figure 2.95: Maximum distance between F_x and F_X .

We can use the function `ks_1samp` from the `scipy.stats` module to compute D_{KS} between any data sample and the CDF of a theoretical distribution. Let's use this function to compute the Kolmogorov–Smirnov distance between normally distributed sample `nsample` and the theoretical distribution $\mathcal{N}(1000, 100)$ from which we generated it.

The function `ks_1samp` expects two arguments: a sample of observations and a cumulative distribution function of the model. We've already defined the computer model `rvN = norm(1000, 100)`, so we have access to the cumulative distribution function F_N under the method `rvN.cdf`.

```
>>> ks_1samp(nsample, rvN.cdf).statistic
0.07815466189999987
```

code
2.7.22

The function `ks_1samp` returns a `results` object with additional information, but we're only interested in the attribute `results.statistic` which corresponds to D_{KS} . The value of the Kolmogorov–Smirnov distance D_{KS} we observe is a relatively small number, which indicates a good fit between the eCDF of `nsample` and the CDF of the model `rvN.cdf`.

A similar calculation allows us to compare the data in `vsample` to the theoretical model `rvV`.

```
>>> ks_1samp(vsample, rvV.cdf).statistic
0.05707018183377044
```

code
2.7.23

Again the observed difference D_{KS} is small, so we conclude there is a good fit.

Finally, we compute the KS distance between `esample` and the theoretical distribution $\text{Expon}(\lambda = 0.2)$, represented as the random variable object `rvE`.

code
2.7.24

```
>>> ks_1samp(esample, rvE.cdf).statistic
0.03597247001342496
```

The fact the observed distance is relatively small tells us there isn't a large difference between `ecdf(esample)` and F_E , so our the random number generation procedure is working correctly.

In contrast, if we were to compute the KS-distance between the exponentially distributed data `esample` and the normal distribution we'll see a very large discrepancy, since the exponential distribution and the normal distribution differ significantly.

```
>>> ks_1samp(esample, norm(5,5).cdf).statistic
0.15871546070321535
```

code
2.7.25

This is quite a big difference, so we conclude that `esample` is not normally distributed.

Note interpreting the D_{KS} values is kind of tricky. Why do we consider 0.078 to be small, but 0.1587 to be large? What happens if we generate smaller or larger samples? We defer further discussion about the interpretation of the KS-distance until Chapter ??, where we'll also discuss other goodness-of-fit metrics.

* * *

You can think of all the above *model-data fit* verifications as alternative measurements of the “difference” between the empirical distribution f_x and the model distribution f_X . The Q-Q plot provides a qualitative measure of the “how far from the quantiles diagonal” distance between the two distributions. Comparing the moments computed from the data and the moments of the theoretical distribution is another approach to check the “fit” between data x and model f_X , which uses the summary statistics (mean, variance, etc.) as the basis of comparison. The Kolmogorov–Smirnov distance uses the cumulative distributions F_x and F_X as the basis of comparison.

2.7.6 Bootstrap sample generation

The concept of *bootstrapping* refers to the process of “reusing” observations from a single sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ to generate “new” observations. This type of “reuse” of sample data for simulation is the backbone of the modern statistics curriculum, and will play an important role in the statistical procedures we'll learn in Chapter ?. Since we've covered random number generation and the empirical distributions in this section, it would be worth giving a quick preview of the bootstrap procedure.

The bootstrap describes the process of sampling from the empirical distribution $f_{\mathbf{x}}$ of the observations $\mathbf{x} = (x_1, x_2, \dots, x_n)$. We can describe the bootstrap observations as the random variable X^* with distribution empirical $f_{\mathbf{x}}$:

$$X^* \sim f_{\mathbf{x}}.$$

Following our usual convention for random variables and their observations, we'll denote a particular observation generated from the bootstrap as x^* .

To generate a bootstrap observation, we can simply select at random one of the x_i s from the data sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$. Recall that the empirical distribution $f_{\mathbf{x}}$ assigns weight $\frac{1}{n}$ to each x_i , so selecting one of the x_i s from \mathbf{x} at random has the same effect as a draw from $f_{\mathbf{x}}$. If want to select multiple observations x_1^* , x_2^* , x_3^* , etc., we repeat the procedure, so in effect we're "sampling with replacement from \mathbf{x} ."

A *bootstrap sample* of size n is defined as a sequence of n observations from the empirical distribution:

$$\mathbf{x}^* = (x_1^*, x_2^*, \dots, x_n^*),$$

where each x_i^* is generated from the empirical distribution $f_{\mathbf{x}}$. The bootstrap sample \mathbf{x}^* is an approximation to what a sample of size n from the distribution f_X would be.

Example: estimating variability from bootstrap samples

Consider the normal random variable N with probability distribution $f_N = \mathcal{N}(1000, 100)$, and a particular sample `nsample` of size $n = 30$ from this distribution.

```
>>> from scipy.stats import norm
>>> rvN = norm(1000, 100)
>>> nsample = rvN.rvs(30)
>>> nsample
[1031.85, 932.04, 779.16, 1059.02, ... 26 more values ... ]
```

code
2.7.26

Suppose Harry has obtained the sample `nsample`, but doesn't know the distribution f_N where the sample came from. Harry can obtain an estimate of the mean μ_N of the unknown distribution f_N by computing the mean from the data sample `nsample`:

```
>>> nsample.mean()
1004.16
```

code
2.7.27

We'll denote Harry's estimate as $\widehat{\mu}_N$, where the hat-on-top is a standard way to denote estimated quantities in statistics. Harry's estimate obtained from the sample is $\widehat{\mu}_N = \mathbf{Mean}(\text{nsample}) = 1004.16$, so Harry knows the mean of the population μ_N is approximately

equal to this value. Harry is aware this estimate is likely “off” by some amount since it was computed from a data sample and not from the real distribution, but it’s the best he can do given the information he has (the $n = 30$ observations in `nsample`).

Suppose Harry wants to also estimate the *variability* in the estimate he computed based on this sample. In other words, if he were to obtain additional random samples of size $n = 30$ from the distribution, and compute the mean from each sample, what kind of sample means would he observe? Unfortunately, Harry can’t generate such samples, since he doesn’t know the distribution f_N . He has to work only with the single sample `nsample` he has on hand.

“No problem,” says Harry to himself, “I’ll use bootstrapping to estimate the variability.” He then runs the following code to generate $B = 10$ bootstrap samples $x_1^*, x_2^*, \dots, x_{10}^*$ from the data in `nsample`, and compute the mean of each bootstrap sample.

```
>>> B = 10
>>> bmeans = []
>>> for k in range(0,B):
        bsample = np.random.choice(nsample, 30)
        bmean = bsample.mean()
        bmeans.append(bmean)
>>> bmeans
[1002.72, 968.09, 995.78, 964.45, 1011.08,
 1013.17, 1027.03, 1041.01, 969.77, 1029.55]
```

code
2.7.28

Each bootstrap sample `bsample` has the same length as the original sample `nsample`. To generate each bootstrap sample, Harry used the function `choice(nsample,30)`, which generates a new sample of length 30 by sampling with replacement from `nsample`.

Harry starts to get an idea about how the mean estimates vary by observing the values like 1002, 968, 995, etc. The maximum deviation between the means computed from the bootstrap samples (`bmeans`) and the mean of the original sample is:

```
>>> max(abs(bmeans - sample.mean()))
39.72
```

code
2.7.29

This means we can expect an estimate of the population mean obtained from a sample of size $n = 30$ to vary by plus-or-minus 40 units. This allows Harry to quantify the uncertainty in the estimate of the population he computed from the sample: `nsample.mean()` = 1004.16. He writes down in his notebook,

$$\widehat{\mu}_N = 1004 \pm 40,$$

as his “best effort” estimate of the mean μ_N of the unknown distribution f_N .

* * *

I know that using bootstrap samples instead of samples from the real distribution seems like cheating, but this approach works out great in practice when trying to obtain uncertainty estimates of quantities computed from a sample.

If you think about it, it makes a lot of sense. If you want to obtain samples from the population but you only have a sample, the best surrogate for the population is the sample you have.

To avoid any possible misunderstandings about this, I want it to be clear that the empirical distribution obtained from a single sample `nsample` is *not* a good approximation of the real distribution f_N . See Figure 2.96 for an illustration. It's not like knowing one sample `nsample` is the same as knowing the distribution f_N overall, but if we're interested in computing some quantity like the sample mean, then computing this quantity based on bootstrap samples will simulate the real variability in samples from f_N .

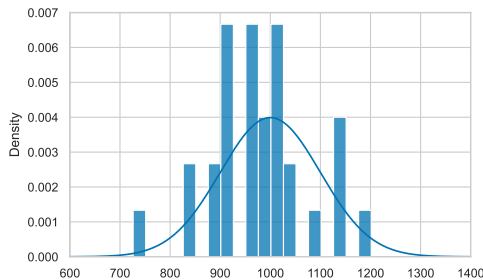


Figure 2.96: Histogram of the empirical distribution obtained from the sample `nsample` of size 30. The superimposed line plot is the probability density function of the distribution f_N .

The “quality” of the bootstrap estimates depends on whether the initial sample is representative of the unknown distribution f_N . When the sample is too small ($n < 30$), the variability observed in it might not be enough to capture the true variability of f_N . Also, if the sample of observed values contains all exceptional values, e.g. particularly large values, then all the bootstrap estimates will be too large as well, since they are all computed by “reusing” the values from the original. We’ll learn more about the bootstrap estimation procedure in Chapter ?? . This was just a preview.

2.7.7 Discussion

Model–data fit as a basis of statistics

The model-data fit question we studied in this section is a very important idea, that we'll see come up again in Chapter ???. In this section we assumed we knew the model's distribution f_X , and we used the model-data fit to check if a random data we generated \mathbf{x} really comes from the model. We can think of all the *data-model fit* checks as computing some kind of “difference” between the empirical distribution $f_{\mathbf{x}}$ and the theoretical distribution f_X , which we can denote as $\text{diff}(\mathbf{x}, f_X)$. Observing large values of $\text{diff}(\mathbf{x}, f_X)$ indicate the random variable procedure we used to generate the observations $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is not working correctly (the data \mathbf{x} does not match the variability of the desired distribution f_X). If our random generation is working correctly, then all the goodness of fit distances $\text{diff}(\mathbf{x}, f_X)$ should be small: data points on the Q-Q plot close to the diagonal, matching moments, and small Kolmogorov–Smirnov distance D_{KS} .

The more common scenario is to have some observations from a real-world process \mathbf{y} and use the model-data fit verification tools to check if the data comes from theoretical distribution family \mathcal{M} . We can describe this scenario as computation of the difference between observed data \mathbf{y} and the whole family of distributions $\text{diff}(\mathbf{y}, \mathcal{M})$. For example, we can ask if a given sample of real-world data observations \mathbf{y} comes from some normal distribution $\mathcal{M} = \mathcal{N}(\mu, \sigma)$, for some parameters μ and σ .

The question of finding the distribution f_Y that “best fits” the dataset \mathbf{y} is one of the central questions in statistics. Specifically, if we assume the distribution f_Y comes from the model family \mathcal{M} , we can rephrase this question as a search for the “best-fit parameters $\hat{\theta}$,” which are defined as the choice of θ that minimize the difference between \mathbf{y} and the distribution $f_Y = \mathcal{M}(\theta)$. Using math notation, the quantity $\hat{\theta}$ is defined as $\hat{\theta} = \min_{\theta} \text{diff}(\mathbf{y}, \mathcal{M}(\theta))$. We'll discuss the different ways of computing estimates $\hat{\theta}$ that “fit” the data in a sample \mathbf{y} in Section ??.

Seeding random number generators

Writing code that involves random number generation can be tricky, since every time you run the code you'll get a different output. The “state” of the random number generator used by your compute has changed, so running the code will produce different random observations.

If you want to make the code execution reproducible, you can

seed the random number generator. If you're using the module `random` as your source of randomness, you need to call the method `random.seed` passing in some arbitrary integer.

```
>>> import random
>>> random.seed(42)
>>> random.random()
0.6394267984578837
```

code
2.7.30

Every time you re-run this code, the output will be the same, because the call to `random.seed(42)` puts the random number generator in the same state.

If you're using any of the NumPy or SciPy models (including all the `scipy.stats` models we discussed), then setting the seed is done differently:

```
>>> import numpy as np
>>> np.random.seed(43)
>>> np.random.rand()
0.11505456638977896
>>> np.random.rand()
```

code
2.7.31

Try running this code block repeatedly and verify the number generated by `np.random.rand()` is always the same.

You don't need to worry about setting the seed in your code, but I mention because might see these types of commands in the notebooks.

Exercises

E2.43 The cumulative distribution function of the random variable $E \sim \text{Expon}(\lambda)$ is $F_E(b) = 1 - e^{-\lambda b}$, for $b \geq 0$. Find the math expression for the inverse cumulative distribution function $F_E^{-1}(q)$.

Hint: Start with the equation $q = 1 - e^{-\lambda b}$ and solve for b .

TODO: graphical question? (show graph of CDF and ask to sample from)

TODO: generate discrete observations from $f_Y(1) = 0.3$, $f_Y(2) = 0.5$, and $f_Y(3) = 0.3$.

simulate random samples X from distribution f_X

TODO use eCDF method

For more exercises, see: https://web.mit.edu/urban_or_book/www/book/chapter7/problems7/

Links

[Empirical distributions]

https://en.wikipedia.org/wiki/Empirical_distribution_function

[Kolmogorov–Smirnov test]

https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test

[Bootstrapping]

[https://en.wikipedia.org/wiki/Bootstrapping_\(statistics\)](https://en.wikipedia.org/wiki/Bootstrapping_(statistics))

2.8 Probability models for random samples

Let's now discuss the properties of random samples like $\mathbf{X} = (X_1, X_2, \dots, X_n)$, which are sequences of n independent copies of the random variable $X \sim f_X$. In particular, we want to study the statistics (function) computed from the random sample \mathbf{X} . An example of a statistic is the mean $\text{mean}(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^n X_i$.

We'll start with some definitions, then describe the notion of the *sampling distribution* of a statistic, and by the end of this section we'll state the *central limit theorem*, which is an important theoretical result about the sampling distribution of the mean computed from random samples.

2.8.1 Definitions

Let's review the probability concepts we have seen previously about sequences of i.i.d. random variables, and define the new concepts.

- X : a random variable with probability distribution f_X
- $\mathbf{X} = (X_1, X_2, \dots, X_n)$: n copies of the random variable X . Each X_i represents an independent copy of the random variable X , and has the same distribution $X_i \sim f_X$. We'll refer to \mathbf{X} as a *random sample*.
- $\mathbf{x} = (x_1, x_2, \dots, x_n)$: a particular sample, which consists of n observations from the distribution f_X .
- *statistic*: any quantity computed from a sample. Examples of statistics include the sample mean $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ and the sample variance $s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$.
- *sampling distribution of a statistic*: the variability of a statistic when computed on a random sample \mathbf{X} . For example, the sampling distribution of the mean is described by the random variable $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$.

Recall the mouthful-of-an-expression *independent, identically distributed* (or *i.i.d* for short), which refers to the sequence of random variables (X_1, X_2, \dots, X_n) , where the distribution of each X_i is identical ($X_i \sim f_X$), and the X_i s are independent. The joint probability distribution for the i.i.d. random sample (X_1, X_2, \dots, X_n) is

$$f_{\mathbf{X}} = f_{X_1 X_2 \dots X_n}(x_1, x_2, \dots, x_n) = \prod_{i=1}^n f_X(x_i).$$

In words, the joint distribution $f_{X_1 X_2 \dots X_n}$ is the product of n copies of the distribution f_X .

2.8.2 Sample statistics

The term *statistic* is used to refer to any quantity computed from a sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$. Examples of statistics include the mean **Mean**, the variance **Var**, and all the other descriptive statistics we learned about in Section ?? . Note we use the term “statistic” to refer to the **function** that computes the quantity of interest, and not the output of the function. The statistics **Mean** and **Var** are functions that take samples as inputs, while **Mean**(\mathbf{x}) and **Var**(\mathbf{x}) are the values of these statistics computed from a particular sample \mathbf{x} .

In this section, we’ll focus our discussion on the sample mean:

$$\mathbf{Mean}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} (x_1 + x_2 + \dots + x_n),$$

which is also denoted as \bar{x} . Let’s write a Python function that computes the mean of a given sample:

```
>>> def mean(sample):
    return sum(sample) / len(sample)
```

code
2.8.1

We assume the input `sample` is any list-like object (a Python list, a NumPy array, or a Pandas series), then use Python builtin function `sum` to compute the summation, and divide by the length of the sample to obtain the output. Note that NymPy arrays and Pandas series have the method `.mean()` you can call to obtain their mean.

Let’s compute the mean of the sample of three observations $\mathbf{x} = [1, 3, 11]$. The math formula tells us $\mathbf{Mean}(\mathbf{x}) = \frac{1+3+11}{3} = \frac{15}{3} = 5$, and we obtain the same answer by calling the function `mean`:

```
>>> mean([1, 3, 11])
5.0
```

code
2.8.2

In the above code example, the *statistic* is the function `mean`, while the number 5.0 is the *value* of the statistic computed from this sample.

The sample mean $\bar{x} = \mathbf{Mean}(\mathbf{x})$ is the most common statistic computed from a sample, but there are many other quantities that we might want to compute from a sample. For example, the variance $s^2 = \mathbf{Var}(\mathbf{x}) = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$, the median $\mathbf{Med}(\mathbf{x}) = F_{\mathbf{x}}^{-1}(\frac{1}{2})$, the position of the third quartile $\mathbf{Q}_3(\mathbf{x}) = F_{\mathbf{x}}^{-1}(0.75)$, or the 90th percentile $F_{\mathbf{x}}^{-1}(0.9)$. All these quantities can be described as computing some function $g(\mathbf{x})$, where g is a function of the form $g : \mathcal{X}^n \rightarrow \mathbb{R}$.

Example 1.1: mean of sample from the uniform distribution

Consider the sequence of observations $\mathbf{u} = (u_1, u_2, \dots, u_n)$, where each u_i is an observation from the standard uniform random variable

$U \sim \mathcal{U}(0,1) = \text{uniform}(0,1)$. The mean of this sample is defined as:

$$\bar{\mathbf{u}} = \mathbf{Mean}(\mathbf{u}) = \frac{1}{n} (u_1 + u_2 + \cdots + u_n) = \frac{1}{n} \sum_{i=1}^n u_i.$$

We'll now run some simulations that generate samples of increasing size and compute the mean from each sample.

```
>>> from scipy.stats import uniform
>>> rvU = uniform(0,1)
>>> mean(rvU.rvs(10))
0.5201367359526748
>>> mean(rvU.rvs(100))
0.459360440874598
>>> mean(rvU.rvs(1000))
0.49846377158147603
>>> mean(rvU.rvs(1000000))
0.5003560391875443
```

code
2.8.3

Note the results we obtain in the above simulations will change every time we run them. Calling the method `rvU.rvs(n)` will return a different set of n observations from the random variable U , so the calculation of the mean statistic will result in a different number.

Observe the mean computed from each sample is close to the mean of the distribution $\mu_U = 0.5$. The means computed from larger samples get closer and closer to μ_U .

Example 1.2: mean of a sample from a normal distribution

We'll now repeat the same procedure for the sample $\mathbf{z} = (z_1, z_2, \dots, z_n)$, where each z_i is an observation from the standard normal random variable $Z \sim \mathcal{N}(0,1) = \text{norm}(0,1)$. The code below computes the sample mean $\mathbf{Mean}(\mathbf{z})$ from samples of different sizes.

```
>>> from scipy.stats import norm
>>> rvZ = norm(0, 1)
>>> mean(rvZ.rvs(10))
-0.26951611032632794
>>> mean(rvZ.rvs(100))
0.04528630589790067
>>> mean(rvZ.rvs(1000))
-0.03750017240797483
>>> mean(rvZ.rvs(1000000))
>>> 0.0006794219838587821
```

code
2.8.4

The mean computed from samples of increasing size seems to get closer and closer to the mean of the distribution $\mu_Z = 0$.

Example 1.3: mean of a sample from an exponential distribution

Let's repeat the exercise a third time, this time generating samples from the exponential random variable $E \sim \text{Expon}(\lambda = 0.2)$.

```
>>> from scipy.stats import expon
>>> lam = 0.2
>>> rvE = expon(0, 1/lam)
>>> mean(rvE.rvs(10))
5.334172048766929
>>> mean(rvE.rvs(100))
4.488399438411224
>>> mean(rvE.rvs(1000))
4.964101264394126
>>> mean(rvE.rvs(1000000))
4.996982168599892
```

We see the same pattern repeat again: the means of larger samples get very close to the mean of the random variable $\mu_E = \frac{1}{\lambda} = 5$.

* * *

As we saw in the above examples, the mean computed from a sample of observations seems to approach the mean of the distribution from which the sample was generated.

This observation can be expressed as a mathematical theorem. The *law of large numbers* (LLN) states that the sample mean $\bar{x} = \text{Mean}(\mathbf{x})$ is approximately equal to the mean of the random variable X , as the sample size n becomes larger and larger.

Theorem (Law of large numbers). *Consider a sample of size n denoted $\mathbf{x}_n = (x_1, x_2, \dots, x_n)$, where each x_i represents an independent draw from the random variable X , which has mean μ_X . Then, the sample mean $\bar{\mathbf{x}}_n = \frac{1}{n} \sum_{i=1}^n x_i$ will converge to the mean of the distribution μ_X :*

$$\bar{\mathbf{x}}_n \rightarrow \mu_X,$$

as the sample size n goes to infinity.

The arrow in the above theorem describes the *limit* behaviour as n becomes larger and larger. Using the math notation for limits, the statement is equivalent to $\lim_{n \rightarrow \infty} \bar{\mathbf{x}}_n = \mu_X$, which is read “the limit of $\bar{\mathbf{x}}_n$ is μ_X .” Note this statement is not saying that the sample mean computed any finite sample size will equal the mean of the distribution. It only describes the limiting behaviour for infinitely large samples.

Don’t worry much about the formal math language or the limit expression: the law of large number is a very intuitive result. If we take larger and larger samples, it makes sense that the mean computed from the sample of observations will approach the mean of the distribution.

This is a very useful thing to happen, since it gives us a practical procedure for estimating the mean of an unknown distribution f_X . If you want to find μ_X , you can simply generate a large enough sample $\mathbf{x} = (x_1, x_2, \dots, x_n)$ from f_X , and you know, thanks to the law of large numbers, that the mean computed from the sample $\bar{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$ will approach the mean μ_X of the distribution f_X .

The law of large numbers tells us the sample mean approaches the mean of the distribution, but it doesn't tell us anything about the *variability* of means computed from random samples $\bar{\mathbf{x}}$. In the next section, we'll develop the vocabulary for describing the variability of statistics.

2.8.3 Sampling distributions of statistics

Recall that a *statistic* is any function computed from a sample. Every statistic corresponds to some function of the form $g : \mathcal{X}^n \rightarrow \mathbb{R}$. In the previous section, we studied the properties of the mean statistic, $g(\mathbf{x}) = \bar{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$, computed from a particular sample of observations $\mathbf{x} = (x_1, x_2, \dots, x_n)$. In this section, we switch our attention to the sequence of random variables $\mathbf{X} = (X_1, X_2, \dots, X_n)$, where each X_i is a copy of the random variable X . We'll refer to $\mathbf{X} = (X_1, X_2, \dots, X_n)$ as a *random sample*.

For example, the mean of the random sample \mathbf{X} is

$$\bar{\mathbf{X}} = \mathbf{Mean}(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^n X_i.$$

Note that $\bar{\mathbf{X}}$ is a random variable, since it is an expression computed based on the random variables (X_1, X_2, \dots, X_n) . The distribution of the mean computed from the random sample \mathbf{X} is called the *sampling distributions of the mean*, and we'll denote it $f_{\bar{\mathbf{X}}}$.

The variance of the random sample is $S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{\mathbf{X}})^2$. The distribution of the random variable S^2 is called the *sampling distributions of the variance*.

In general, given any statistic we might want to compute from a sample (a function g of the form $g : \mathcal{X}^n \rightarrow \mathbb{R}$), the "sampling distribution of g " describes the variability of the quantity $g(\mathbf{X})$, which is the statistic computed from a random sample $\mathbf{X} = (X_1, X_2, \dots, X_n)$.

Note the difference between $g(\mathbf{x})$ (a number) and $g(\mathbf{X})$ (a random variable). Calculating the statistic g on a particular sample \mathbf{x} (a sequence of observations) results in a single number: the value of the statistic function computed from that sample. In contrast, the sampling distribution $g(\mathbf{X})$ represents the calculation of the statistic g over *all possible* sequences $\mathbf{X} = (X_1, X_2, \dots, X_n)$.

One approach to study the properties of the sampling distribution $g(\mathbf{X})$ is to use simulation. We can generate many (thousands) random samples \mathbf{x} , compute the value of the statistic function on each sample $g(\mathbf{x})$, then plot a histogram of these observations $g(\mathbf{x})$. We'll now illustrate this procedure in the following examples.

Example 2.1: sampling distribution of $\text{Mean}(\mathbf{U})$

Uma just received a notification that her assignment for her probability class is due in one hour. She quickly goes through the first three questions of the assignment since she remembers most of the material, but then she reaches a question asking her to “compute the mean and the variance of the sampling distribution of $\text{Mean}(\mathbf{U})$ ” and she’s not 100% sure what to do. She vaguely remembers hearing about sampling distributions in class, but not the details. She could try to look up the answer online, but given how little time is left to finish the assignment, she feels there is no time to “learn” all the theory, and instead the right thing to do is to open a Jupyter notebook and just try to solve the problem on her own. The teacher said Python is powerful. Let’s see how powerful it is.

The exact wording of the questions is: “Find the mean and the variance of the sampling distribution of $\text{Mean}(\mathbf{U})$, where $\mathbf{U} = (U_1, U_2, \dots, U_{30})$ is a random sample that consists of 30 independent copies of the standard uniform random variable $U \sim \mathcal{U}(0, 1)$.”

“Okay Python,” she says to herself. “You and me. We’ve got 36 minutes left to figure this shit out. Let’s do this!”

Uma starts by importing the uniform model from the module `scipy.stats`, and since the question talks about the standard uniform random variable $U \sim \mathcal{U}(0, 1)$, she creates a computer model `rvU` that represents this random variable.

```
>>> from scipy.stats import uniform
>>> rvU = uniform(0,1)
```

code
2.8.6

The question is asking something about the properties of random samples of size $n = 30$ from the random variable `rvU`. Uma is still not sure what $\mathbf{U} = (U_1, U_2, \dots, U_{30})$ means (why the capitals?), but she knows the lowercase version $\mathbf{u} = (u_1, u_2, \dots, u_{30})$ corresponds to generating a simple of size $n = 30$ from the random variable `rvU`. Uma knows this is possible by calling the method `rvU.rvs(30)`, as shown in the code below.

```
>>> n = 30
>>> usample = rvU.rvs(n)
>>> usample
[0.8624, 0.3383, 0.1980, ... 27 more numbers ... ]
```

code
2.8.7

The assignment question is asking something about the **Mean** statistic, so Uma copy-pastes the definition of the Python function `mean` from code block 2.8.1 into her assignment notebook:

```
>>> def mean(sample):
        return sum(sample) / len(sample)
```

code
2.8.8

Uma can now compute the value of the statistic **Mean** from of any sample. She decides to try this out on the sample of random observations $\mathbf{u} = \text{usample}$ that she just generated.

```
>>> mean(usample)
0.5887109207913922
```

code
2.8.9

The question is asking to find the sampling distribution of **Mean(U)**, so Uma feels she's on the right track. She knows how to generate random samples of n observations \mathbf{u} by calling `rvU.rvs(30)`, and she knows how to compute the mean of a particular sample **Mean(u)** using the function `mean`. But how do we go from **Mean(u)** to **Mean(U)**?

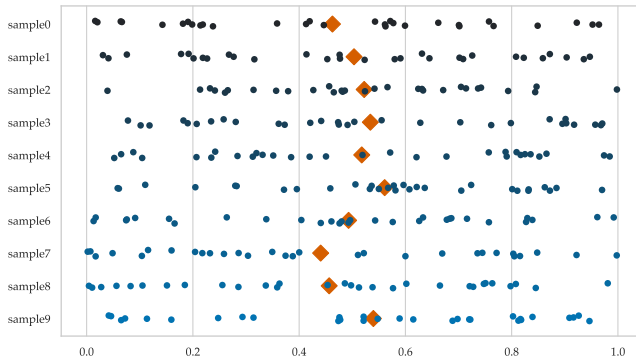


Figure 2.97: Scatter plots of 10 samples of size $n = 30$ from the standard uniform distribution $\mathcal{U}(0,1)$. The sample mean computed from each sample is indicated with the diamond marker.

To get an idea of the variability of the individual observations U_i , Uma generated 10 samples of size $n = 30$ from \mathbf{U} , and creates a combined strip plot of the samples, as shown in Figure 2.97. She also computes the mean for each sample and display it as a diamond marker in the figure.

Then it finally clicks—the **sampling distribution of Mean(U) describes the variability of the diamond markers** in Figure 2.97. The individual observations u_i are uniformly distributed over the interval $[0,1]$, but the sample means are all concentrated close to the value 0.5, with much less variability.

In order to quantify the variability of the sampling distribution **Mean(U)**, Uma decides to generate $N = 1000$ random samples $\mathbf{u}_1,$

$\mathbf{u}_2, \mathbf{u}_3, \dots, \mathbf{u}_N$, and compute the mean $\bar{\mathbf{u}}_i = \mathbf{Mean}(\mathbf{u}_i)$ from each sample, to obtain the list of observations $[\bar{\mathbf{u}}_2, \bar{\mathbf{u}}_2, \bar{\mathbf{u}}_3, \dots, \bar{\mathbf{u}}_N]$, which are corresponds to 1000 samples from $\mathbf{Mean}(\mathbf{U})$.

Uma decides to use a Python for-loop to generate the 1000 random samples, and record the observations $\bar{\mathbf{u}}_i$ in a list named `xbars`. She quickly comes up with the following simulation code.

```
>>> n = 30      # sample size
>>> N = 1000    # number of samples to generate
>>> ubars = []
>>> for i in range(0,N):
>>>     usample = rvU.rvs(n)
>>>     ubar = mean(usample)
>>>     ubars.append(ubar)
```

code
2.8.10

After running this code, the list `ubars` contains 1000 observations from the sampling distribution, which she can now visualize by generating a histogram and a strip plot of the values.

```
>>> sns.histplot(ubars, color="r")
>>> sns.scatterplot(x=ubars, y=-5, color="r", marker="D")
```

code
2.8.11

The result is shown in Figure 2.98.

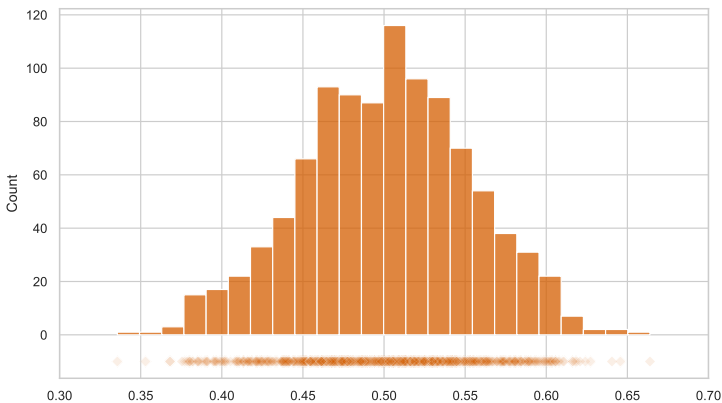


Figure 2.98: An approximation to the sampling distribution of $\mathbf{Mean}(\mathbf{U})$ computed from $N = 1000$ samples of size $n = 30$ taken from the standard uniform distribution $\mathcal{U}(0,1)$. Most of the sample means computed from these samples fall close to the distribution mean $\mu_U = 0.5$.

By inspecting Figure 2.98 Uma starts to get an intuitive understanding of the sampling distribution of $\mathbf{Mean}(\mathbf{U})$ (also denoted $f_{\bar{\mathbf{U}}}$). She notes the distribution is centred around 0.5 and has a standard deviation of roughly 0.05 around the mean. These are the numbers the assignment is asking her to compute!

To obtain precise numerical estimates of the mean and the standard deviation of the sampling distribution, Uma decides to compute the mean and standard deviation of the list `ubars`.


```
>>> np.mean(ubars), np.std(ubars)
(0.5000180761308989, 0.05211860142360233)
```

The mean of the sampling distribution is $\mu_{\bar{U}} = 0.5$ and its standard deviation is $\sigma_{\bar{U}} = 0.052$. She quickly enters these answers to the question and submits the online assignment. The server responds with a positive feedback—the answers are correct.

Looking at the clock, Uma realized there are still 30 minutes left before the deadline. Maybe the teacher is right, using Python to run simulations really is very useful for solving probability questions.

General-purpose generator of sampling distributions

The ability to generate observations from the sampling distribution of a statistics is a very useful skill to have. Let's take a moment to generalize Uma's code so we can reuse it in other situations. When discussing sampling distributions, we need to know three things:

- The sample size n (an integer).
- The random variable X that is used to form the random sample $\mathbf{X} = (X_1, X_2, \dots, X_n)$.
- The statistic of interest (a function computed from samples).

In the case of Uma's calculations we saw in Example 2.1, the sample size was $n = 30$, the random variable was $U \sim \mathcal{U}(0,1)$, and the statistic she wanted to calculate from each sample was **Mean**.

We can obtain a general-purpose sampling distribution generator by turning Uma's code into a Python function `gen_sampling_dist` that takes three arguments: `rv` the model for the random variable (an instance of one of the computer models from `scipy.stats`), the statistic of interest (a Python function), and the sample size.

```
>>> def gen_sampling_dist(rv, statfunc, n, N=1000):
    stats = []
    for i in range(0, N):
        sample = rv.rvs(n)
        stat = statfunc(sample)
        stats.append(stat)
    return stats
```

code
2.8.13

The function also takes an optional parameter N , which controls the number of observations from the sampling distribution will be generated. The default value $N = 1000$ is reasonable for most situations—with 1000 data points, we can draw pretty accurate histograms.

We can reproduce Uma's calculations by calling the function `gen_sampling_dist` with the following choice of arguments: set `rv` to the computer model `rvU = uniform(0,1)`, set `statfunc` to the function `mean`, and set `n` to the value 30.

```
>>> ubars = gen_samp_dist(rvU, statfunc=mean, n=30)
>>> np.mean(ubars), np.std(ubars)
(0.5034057868782315, 0.052257079327563884)
```

Note the mean and the standard deviations we obtain from the list of observations `ubars` is very similar to the numbers `Uma` obtained, but not identical. This is to be expected when running simulations, there will always be small differences, but the answers are very close.

Example 2.2: sampling distribution of $\text{Mean}(\mathbf{Z})$

Uma's classmate Zach got a similar question on his assignment, but asking to compute the mean and the standard deviation of the sampling distribution of $\text{Mean}(\mathbf{Z})$, where $\mathbf{Z} = (Z_1, Z_2, \dots, Z_{30})$ is a random sample of size 30 from the standard normal $Z \sim \mathcal{N}(0, 1)$.

Zach starts by creating the computer model `rvZ = norm(0, 1)` for the standard normal random variable $Z \sim \mathcal{N}(0, 1)$.

```
>>> from scipy.stats import norm
>>> rvZ = norm(0, 1)
```

code
2.8.15

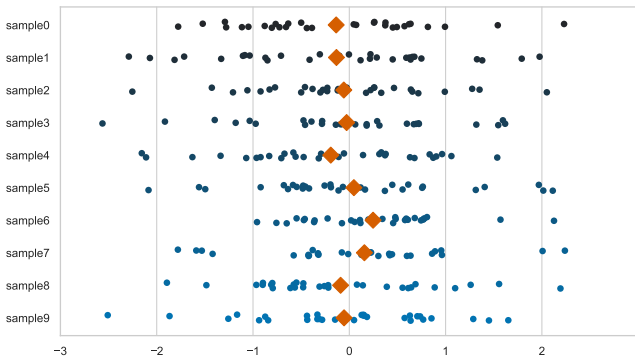


Figure 2.99: Scatter plots of ten samples of size $n = 30$ from the standard normal distribution $Z \sim \mathcal{N}(0, 1)$. The sample mean computed from each sample is indicated with the diamond marker.

Figure 2.99 shows strip plots of ten samples of size $n = 30$, generated by calling `rvZ.rvs(30)`, with the diamond shapes indicating the values of $\text{Mean}(\mathbf{z}_i)$ computed from each sample \mathbf{z}_i .

In order to generate observations from the sampling distribution of $\text{Mean}(\mathbf{Z})$, Zach can call the function `gen_sampling_dist`, passing in the random variable object `rvZ` as the first argument, the function mean as the `statfunc`, and sample size $n = 30$ as the third argument.

```
>>> zbars = gen_sampling_dist(rvZ, statfunc=mean, n=30)
```

code
2.8.16

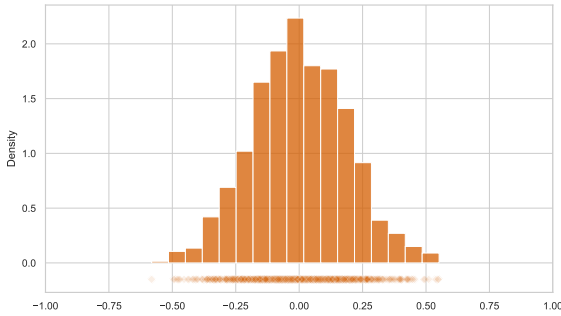


Figure 2.100: Sampling distribution of the mean computed from $N = 1000$ samples of size $n = 30$ taken from the standard normal distribution $\mathcal{N}(0, 1)$. Most of the sample means fall close to the mean of the distribution $\mu_Z = 0$.

Figure 2.100 shows the histogram and scatter plot of the observations \mathbf{zbars} . Similar to what we saw in Example 2.1, the sampling distribution of the mean has the shape of a normal distribution. Using visual inspection, Zach estimates the mean of the sampling distribution is zero, and its standard deviation is roughly 0.2.

To obtain more precise estimates of the mean and the standard deviation of the sampling distribution, Zach computes the mean and the standard deviation of the values in the list \mathbf{zbars} :

```
>>> np.mean(zbars), np.std(zbars)
(0.0019901929294202933, 0.186121326960664)
```

code
2.8.17

He then uses the answers $\mu_{\bar{Z}} = 0$ for the mean and $\sigma_{\bar{Z}} = 0.186$ for the standard deviation to answer the assignment question.

* * *

Meanwhile, at the other end of town, Zoe got the same question as Zach on her assignment: compute the mean $\mu_{\bar{Z}}$ and the standard deviation $\sigma_{\bar{Z}}$ of the sampling distribution of $\mathbf{Mean}(\mathbf{Z})$. Zoe has a math background, and remembers reading about a similar type of calculation in Example 7, back in Section 2.5 (see page 158). She recognizes the random variable A in Example 7 is exactly the same as the quantity $\mathbf{Mean}(\mathbf{Z})$, and she knows how to compute the mean and the variance of the random variable using math equations.

The mean $\mu_{\bar{Z}}$ of the sampling distribution $\mathbf{Mean}(\mathbf{Z})$ is the same as computing the mean of the random variable A :

$$\mu_A = \mathbb{E}_A[A] = \mathbb{E}_Z \left[\frac{1}{n} \sum_{i=1}^n Z_i \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}_{Z_i}[Z_i] = \frac{1}{n} [n \cdot \mu_Z] = \mu_Z.$$

Zoe used the linearity of expectations to obtain the third equation, which is allowed since the random variables Z_i are independent. The fourth equation follows because the Z_i s are identical copies of the random variable Z . Knowing the mean of the standard normal is $\mu_Z = 0$, tells her the first part of the answer is $\mu_{\bar{Z}} = 0$.

The variance $\sigma_{\bar{Z}}^2$ of the sampling distribution \bar{Z} is the same as the variance of the random variable A :

$$\sigma_A^2 = \text{var}(A) = \text{var}\left(\frac{1}{n} \sum_{i=1}^n Z_i\right) = \frac{1}{n^2} \text{var}\left(\sum_{i=1}^n Z_i\right) = \frac{1}{n^2} \cdot n \cdot \text{var}(Z) = \frac{\sigma_Z^2}{n}.$$

Zoe used the variance formula $\text{var}(mX) = m^2 \cdot \text{var}(X)$ to obtain the third equation. The fourth equation follows because the random variables Z_i are independent. Knowing the standard deviation of the random variable Z is $\sigma_Z = 1$, she quickly calculates the desired answer $\sigma_{\bar{Z}} = \sqrt{\frac{\sigma_Z^2}{n}} = \sqrt{\frac{1}{30}} \approx 0.18257$ using her calculator. She submits this answer and gets full points on this question, feeling happy about how her math skills have helped her.

Example 2.3: sampling distribution of Mean(E)

Erica got a different version of this question on her assignment, asking her to compute the properties of the sampling distribution of **Mean(E)**, where $\mathbf{E} = (E_1, E_2, \dots, E_{30})$ is a random sample of size 30 from the exponential random variable $E \sim \text{Expon}(\lambda = 0.2)$.

She knows she needs to build a computer model `rvE` for the random variable E , so she consults the SciPy documentation about the computer model `scipy.stats.expon`, in order to know what parameters she must specify. She realizes she needs to set the `loc` (location) parameter to zero, and the scale parameter to $\frac{1}{\lambda}$ to obtain the appropriate computer model.

```
>>> from scipy.stats import expon
>>> lam = 0.2 # lambda
>>> scale = 1/lam
>>> rvE = expon(0, scale)
```

code
2.8.18

Figure 2.101 shows a ten samples of size $n = 30$, that Erica obtained by calling `rvE.rvs(30)`. The diamond shapes indicating the mean computed from each sample.

She then generates 1000 observations from the sampling distribution of **Mean(E)** by calling the function `gen_sampling_dist` and passing in the computer model `rvE` as the first argument.

```
>>> ebars = gen_sampling_dist(rvE, statfunc=mean, n=30)
```

code
2.8.19

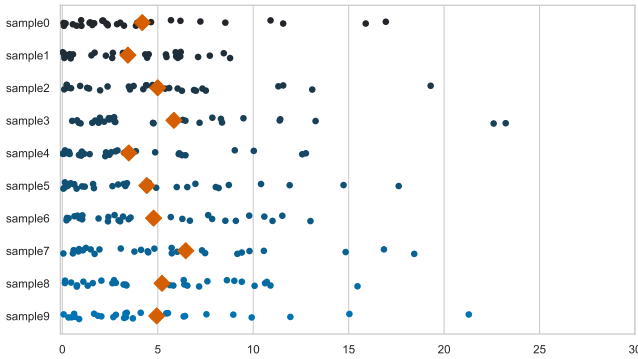


Figure 2.101: Scatter plots of ten samples of size $n = 30$ from the exponential distribution $\text{Expon}(\lambda = 0.2)$. The sample mean computed from each sample is indicated with the diamond marker.

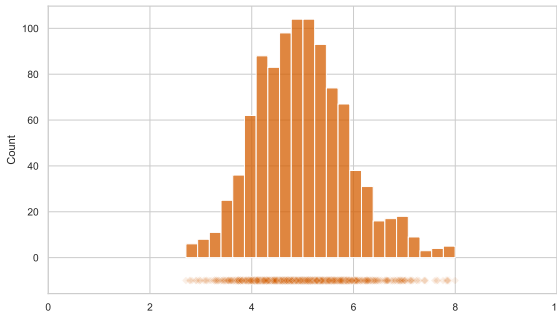


Figure 2.102: Sampling distribution of the mean computed from $N = 1000$ samples of size $n = 30$ taken from the exponential distribution $\text{Expon}(\lambda = 0.2)$. Most of the sample means computed from these samples fall close to the distribution mean $\mu_E = \frac{1}{\lambda} = 5$.

To visualize the distribution of the observations obtained in `ebars`, Erica plots the histogram and strip chart shown in Figure 2.100. She observes that the shape of the sampling distribution of $\text{Mean}(\mathbf{E})$ kind of looks like a Gaussian distribution, except for some extra values in the right tail.

She computes the mean and the variance of the sampling distribution using the code:

```
>>> np.mean(ebars), np.std(ebars)
(5.02490198427453, 0.9335571543972068)
```

code
2.8.20

She provides the answers $\mu_{\mathbf{E}} = 5.02$ and $\sigma_{\mathbf{E}} = 0.933$ for this question and gets full marks on it.

Note that the sampling distribution of the mean computed in all three of the above examples turns out to have a normal shape. We would expect this to happen for samples from the normal distribution f_Z , it makes sense for the average of n observations from the normal distribution to also be normally distributed. The distributions f_U and f_E are not normal, yet the sampling distributions we obtained from them turn out to be normally distributed. What is up with that? How come the normal distribution pops up out of nowhere? This is not a coincidence, but in fact an important theorem of probability theory, which we'll discuss next.

2.8.4 Central limit theorem

The central limit theorem (CLT) tells us that the sampling distribution of the mean computed from i.i.d. samples of *any* distribution is approximately normally distributed.

Theorem (Central limit theorem). *Consider a random sample of size n denoted $\mathbf{X} = (X_1, X_2, \dots, X_n)$, where each X_i represents an independent draw from the random variable X . Let μ_X denote the mean of the random variable X , and let σ_X denote the standard deviation of X . Then the sampling distribution of the mean $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ will converge to a normal distribution:*

$$\bar{X} \rightarrow \mathcal{N}\left(\mu_X, \frac{\sigma_X}{\sqrt{n}}\right),$$

as the sample size n goes to infinity.

We already knew from the law of large numbers that the mean of the sampling distribution \bar{X} will approximately equal the mean of the random variable μ_X , so this is not new.

What is new is that the central limit theorem tells us the “shape” of the sampling distribution of the mean **Mean**(\mathbf{X}) will be a normal distribution. This is an interesting fact by itself: no matter what model distribution f_X we start from (not necessary normal), the sample means will be normally distributed.

Moreover, the central limit theorem gives us a math formula for the standard deviation of the sampling distribution as a function of the sample size n :

$$\sigma_{\bar{X}} = \frac{\sigma_X}{\sqrt{n}}.$$

Equivalently, we can state the predictions of the central limit theorem in terms of variances: the variance of the sampling distribution of the mean \bar{X} computed from samples of size n is equal to $\frac{\sigma_X^2}{n}$.

What exactly do we mean when we say “as n goes to infinity” in the CLT? I used this informal expression in order to avoid getting into the technical discussions, which are not required for using the central limit theorem in practice. The CLT is *exactly* true only in the limit of infinitely large sample size n . For finite-size samples, $n = 10$, $n = 30$, $n = 100$, etc., the statement of the CLT is only approximately true $f_{\bar{X}} \approx \mathcal{N}\left(\mu_X, \frac{\sigma_X}{\sqrt{n}}\right)$, with the accuracy of this approximation increasing as n increases. In practice, we often use the criterion $n \geq 30$ as a rule of thumb for when we start to consider the approximation to be accurate, but there is no real universal threshold that we can apply for all situations. For different models, the convergence could be faster (thus allowing small sample sizes like $n = 10$) or slower (requiring larger sample sizes like $n = 100$).

Roughly speaking, the closer the distribution f_X is to normal, the smaller the sample size needs to be for the approximation to be acceptably accurate. Unless the distribution f_X is multimodal or severely skewed, a sample size $n \geq 30$ is usually sufficient for the approximation to hold.

Let’s look at some examples in which we apply the central limit theorem to different distributions and different samples sizes.

Example 3.1: CLT for samples from the uniform distribution

Figure 2.103 shows strip plots for random samples of different sizes generated from the standard uniform random variable $U \sim \mathcal{U}(0, 1)$. Note that the variability of the sample means decreases as the sample size increases.

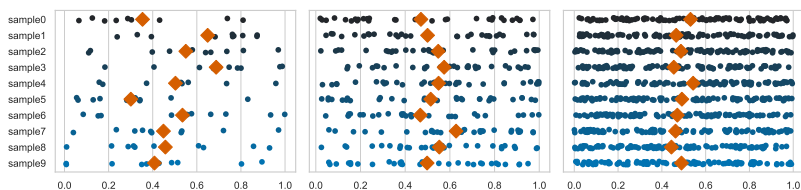


Figure 2.103: Strip plots of samples of different size $n = 10$, $n = 30$, and $n = 100$ from the standard uniform distribution $\mathcal{U}(0, 1)$. The sample mean of each sample is indicated with the diamond marker.

Figure 2.104 shows a side-by-side comparison of the sampling distributions of $\text{Mean}(U)$, computed from random samples of different size. The left panel shows the histogram of samples `ubars10` which contains observations from sampling distribution of the mean computed samples of size $n = 10$. We see there is a reasonable agreement between the histogram and the theoretical model suggested

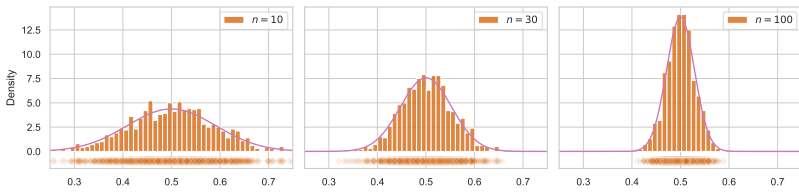


Figure 2.104: Comparison of the sampling distribution of the mean computed from samples of size $n = 10$, $n = 30$, and $n = 100$ from the standard uniform $\mathcal{U}(0, 1)$. The purple line indicates the mathematical model obtained by applying the central limit theorem.

by the central limit theorem $\mathcal{N}(0.5, \frac{\sigma_U}{\sqrt{10}})$. For sample sizes of size $n = 30$ as shown in the middle panel, In the middle panel we see the same situation repeated with samples of size $n = 30$, and the right panel the sampling distribution obtained from samples of size $n = 200$. Note how the variability of the sample mean estimates decreases when we take larger samples, and the model predicted by the central limit theorem correctly captures this effect.

Let's compute the standard deviation of the sampling distributions for the different sizes.

```
>>> np.std(ubars10), np.std(ubars30), np.std(ubars100)
(0.0882, 0.0522, 0.0283)
```

code
2.8.21

The predictions of the central limit theorem for samples of these sizes are as follows.

```
>>> from math import sqrt
>>> rvU.std()/sqrt(10), rvU.std()/sqrt(30), rvU.std()/sqrt(100)
(0.0912, 0.0527, 0.0288)
```

code
2.8.22

Comparing the two sets of numbers, we see there is some disagreement between the theoretical prediction of the CLT for samples of size $n = 10$, but for samples of size $n = 30$ and $n = 100$ the predictions of the CLT are a very good match to the standard deviations we calculated through simulation.

Example 3.2: CLT for samples from the normal distribution

Let's repeat the exercise using samples from the standard normal distribution $Z \sim \mathcal{N}(0, 1)$. Figure 2.105 shows strips plots of a few samples, while Figure 2.106 shows the sampling distribution.

The standard deviations of the sampling distributions we obtained through simulation are:

```
>>> np.std(zbars10), np.std(zbars30), np.std(zbars100)
(0.3181, 0.1854, 0.0999)
```

code
2.8.23

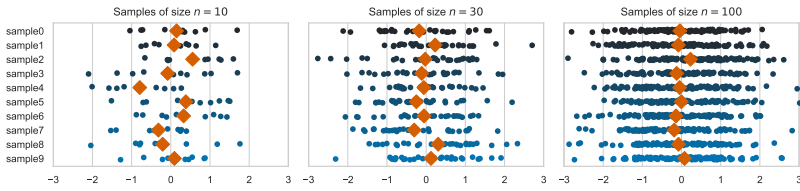


Figure 2.105: Strip plots of samples of different size $n = 10$, $n = 30$, and $n = 100$ from the standard normal distribution $\mathcal{N}(0, 1)$. The sample means are indicated as diamond markers.

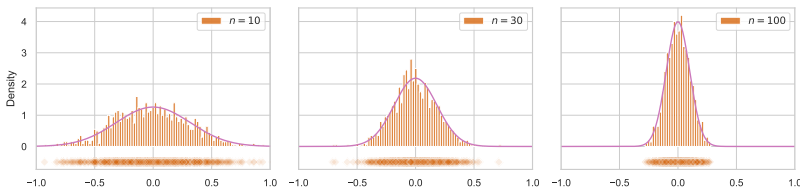


Figure 2.106: Comparison of the sampling distribution of the mean computed from samples of size $n = 10$, $n = 30$, and $n = 100$ from the standard normal distribution $\mathcal{N}(\mu = 0, \sigma = 1)$.

Let's compare these observations from the theoretical prediction of the CLT:

```
>>> from math import sqrt
```

code
2.8.24

```
>>> rvZ.std()/sqrt(10), rvZ.std()/sqrt(30), rvZ.std()/sqrt(100)
(0.3162, 0.1825, 0.1)
```

We see there is good agreement between the result of our simulation and the prediction of the CLT, even for the small sample size $n = 10$.

Example 3.3: CLT for samples from the exponential distribution

We'll now repeat the sampling distribution visualizations a third time, this time applying it to using samples from the exponential random variable $E \sim \text{Expon}(\lambda = 0.2)$.

The standard deviations obtained through simulation are

```
>>> np.std(ebars10), np.std(ebars30), np.std(ebars100)
(1.660126019569815, 0.9051707388579937, 0.522425074266255)
```

code
2.8.25

The CLT predictions are

```
>>> rvE.std()/sqrt(10), rvE.std()/sqrt(30), rvE.std()/sqrt(100)
(1.581, 0.9128, 0.5)
```

code
2.8.26

The approximation is waaaaay off when $n = 10$, not bad for $n = 30$, and starts to become good at $n = 100$.

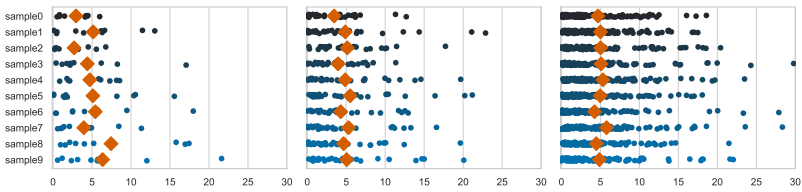


Figure 2.107: Strip plots of samples of different size $n = 10$, $n = 30$, and $n = 100$ from the exponential distribution $\text{Expon}(\lambda = 0.2)$, with the sample means indicated as diamonds.

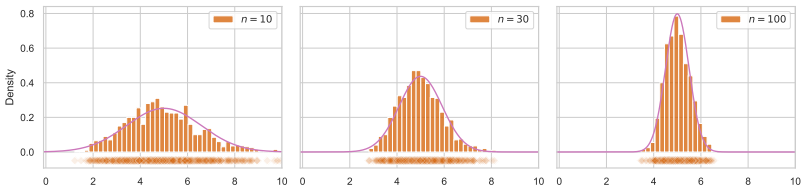


Figure 2.108: Comparison of the sampling distribution of the mean computed from samples of size $n = 10$, $n = 30$, and $n = 100$ from the exponential distribution $\text{Expon}(\lambda = 0.2)$.

2.8.5 Discussion

The topics introduced in this section are some of the most important tools we'll use in statistics. It's essential that you understand the general concept of a sampling distribution, and how to generate observations from the sampling distribution. Make sure you understand how the function `gen_sampling_dist` works (see code 2.8.13). This general-purpose procedure will come in handy later in the book.

Going deeper into the math

Readers who are interested in digging deeper into the two theorems we presented, LLN and CLT, will need to consult some more advanced books to learn about the details.

When talking about the LLN, mathematicians can describe the exact nature of the convergence of \overline{X}_n to μ_X . The *weak law of large numbers* states that $\lim_{n \rightarrow \infty} \Pr(|\overline{X}_n - \mu_X| < \varepsilon) = 1$, for all $\varepsilon > 0$. The *strong law of large numbers* states $\Pr(\lim_{n \rightarrow \infty} \overline{X}_n = \mu_X) = 1$, which means \overline{X}_n will become exactly μ_X . This paper[<https://doi.org/10.3150/12-BEJSP12>] provides an interesting historical overview of the different version of LLN that appeared throughout the years.

Similarly, there are different formal math statements for the CLT. Learning those details requires measure theoretic math tools, which

is out of scope for this book. We'll focus on the “write a for-loop to run a simulation” approach, rather than try to prove mathematically precise theorems.

CLT is the basis for all analytical approximations in statistics

The central limit theorem is a very powerful result, that you can use whenever we're analyzing the mean statistic computed from random samples, which is a very common scenario in statistics. Indeed, we could say half the analytical approximations formulas normally covered in the STATS 101 course are consequences of the CLT. It wouldn't be an exaggeration to say the CLT is a pillar of statistics.

According to the CLT, the variability of the sampling distribution decreases as the sample size n increases. In some sense, the CLT is the reason why statistics works. We can use the properties samples to estimate the parameters of the population, and our estimates get more accurate if we use large samples.

We'll talk about this “everything is approximately normal” phenomenon at length in Section ??, and later in Section ??.

Exercises

simulate random samples X from distribution f_X

numerically compute expectations of sample statistics (and compare with theory)

compute sampling distribution for different statistics using simulation

verify/observe the CLT for various distributions f_X
(what happens when $n = 10$ and $n = 15$ from exponential... is the sampling dist. of mean still normal?)

Links

[Law of large numbers on Wikipedia]

https://en.wikipedia.org/wiki/Law_of_large_numbers

[Central limit theorem on Wikipedia]

https://en.wikipedia.org/wiki/Central_limit_theorem

[*Seeing theory* book by Devlin, Guo, Kunin, and Xiang]

<https://seeing-theory.brown.edu/doc/seeing-theory.pdf>

Conclusion

If you want to have a good time in STATS, you need to make sure you understand probability theory and are familiar with the various prob. distributions we discussed. These distributions will be your basic building blocks you'll use repeatedly in the rest of the book. The best way to do that is to solve practice problems and apply the concepts to various situations, simple and complex. Reading is not enough—you need get “hands on” experience with probability distributions.

How convenient is it that the next section has exactly a bunch of probability practice problems. Go grab a caffeinated beverage of your choosing, and sit down to try some of these.

2.9 Probability problems

Intro/motivational text...

P2.1 The *addition rule* of probability states that

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A \cap B),$$

for any two sets of outcomes A and B . The symbol \cup describes the *union* of two sets, which means all elements that are either A or B . The symbol \cap describes the *intersection* of the two sets, meaning the elements that are in A and B .

Verify the addition rule applies to the die roll random variable D , and the sets of outcomes $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$, by computing the probabilities of all terms in the above equation.

2.9.1 Simple probability problems

To better understand random variables and probability distributions, you need to practice using these concepts to solve real-world problems. It just so happens there are some practice problems on this very topic in this section—how convenient is that? Don't skip them!

P2.1 Given a random variable X with three possible outcomes $\{1, 2, 3\}$ and probability distribution $f_X = (p_1, p_2, p_3)$, prove that $p_1 \leq 1$.

Hint: Use the Kolmogorov's axioms and build a proof by contradiction.

P2.2 The probability of heads for a fair coin is $p = \frac{1}{2}$. The probability of getting heads n times in a row is given by the expression p^n . What is the probability of getting heads four times in a row?

P2.3 You have a biased coin that lands on heads with probability p , and consequently lands on tails with probability $(1 - p)$. Suppose you want to flip the coin until you get heads. Define the random variable N as

the number of tosses required until the first heads outcome. What is the probability mass function $P_N(n)$ for success on the n^{th} toss? Confirm that the formula is a valid probability distribution by showing $\sum_{n=1}^{\infty} P_N(n) = 1$.

Hint: Find the probabilities for cases $n = 1, 2, 3, \dots$ and look for a pattern.

P2.4 The probability mass function for the geometric distribution with success probability p is $f_X(x) = (1-p)^{x-1}p$, where X describes the number of trials until the first success. Compute the expected value $\mathbb{E}_X[X]$.

Hint: The formula for the sum of the geometric series is $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$, and taking its derivative with respect to r gives $\sum_{k=0}^{\infty} k r^{k-1} = \frac{1}{(1-r)^2}$.

P2.5 A mathematician walks over to a roulette table in a casino. The roulette wheel has 101 numbers: 50 are black, 50 are red, and the number zero is green. If the mathematician bets \$1 on black and the roulette ball stops on a black number, the payout is \$2, otherwise the bet is lost. Calculate the expected gains from playing this game, and determine whether it's worth playing.

P2.6 Consider the following variation of the six-sided die game. You pay \$1 to play one round of the game and the payout for the game is as follows. If you roll a \square , a \square , or a \square , you win nothing. If you roll a \square or a \square , you win \$1. If you roll a \square , you win \$5. Should you play this game?

P2.7 Show that variance of a random variable X with distribution f_X is given by the formula $\text{var}(X) = \sum_{x \in \mathcal{X}} x^2 f_X(x) - \mu_X^2$.

Hint: Start from the definition $\text{var}(X) \equiv \mathbb{E}_X[(X - \mu_X)^2]$ and simplify it.

P2.8 Perform the simulation analysis (proportion of wrong decisions made and or errors in CI intervals) if statistical test uses z scores instead of t statistic, as a function of: (1) effect size (2) sample size (d.f.)

2.9.2 Discrete distributions problems

Intro/motivational text...

P2.1 Compute the variance of the random variable $X \sim \text{Binom}(n, p)$.

Hint: Use the relation to the sum of independent Bernoulli variables, and use the variance properties for independent variables.

P2.2 Compute the variance of the random variable $X \sim \text{Binomial}(n, p)$ whose distribution is $p_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$, for $k \in \{0, 1, \dots, n\}$.

Hint: Start from the formula $\sigma^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, then add and subtract $\mathbb{E}[X]$ and rewrite as $\sigma^2 = \mathbb{E}[X(X-1)] + \mathbb{E}[X] - \mathbb{E}[X]^2$.

P2.3 Consider the random variable X that describes the number of events during the time interval $[0, T]$ which has length T . We assume, a priori, that the number of successes per unit length has an

average of r so the expected value of successes over the time period of length T is $\mathbb{E}[X] = rT$, which we'll give a new name λ .

Show that gives the Poisson distributoin.

P2.4 Compute the variance of the random variable $X \sim \text{Geometric}(p)$ with probability mass function $p_X(k) = (1-p)^{k-1}p$, for $k \in \{1, 2, \dots\}$.

Hint: Start from the formula $\sigma^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

P2.5 TODO: Replace $k \in r \dots$ to $x \in 0 \dots$;

Compute the variance of $X \sim \text{NegativeBinomial}(r, p)$ whose distribution is $p_X(k) = \binom{k-1}{r-1} (1-p)^{k-r} p^r$, for $k \in \{r, r+1, r+2, \dots\}$.

Hint: Use the formula $\sigma^2 = \mathbb{E}[X(X-1)] + \mathbb{E}[X] - \mathbb{E}[X]^2$ as in P2.2.

P2.6 Replace k with x

Replace n with $a+b$

Replace K with a

Replace N with n

Compute the variance of the random variable $X \sim \text{Hypergeometric}(n, K, N)$ with probability mass function $f_X(k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}}$, for

Hint: Use the formula $\sigma^2 = \mathbb{E}[X^2] - \mathbb{E}[X]^2$.

P2.7 Compute the variance of the random variable $X \sim \text{Poisson}(\lambda)$ with probability mass function $p_X(k) = \frac{(\lambda)^k e^{-\lambda}}{k!}$, for $k \in \{0, 1, 2, \dots\}$.

Hint: Use the formula $\sigma^2 = \mathbb{E}[X(X-1)] + \mathbb{E}[X] - \mathbb{E}[X]^2$.

P2.8 Suppose $Z = \sum_{i=1}^n a_i Z_i$ is a linear combination of independent random variables each having means μ_i . Show that $\mathbb{E}[Z] = \sum_{i=1}^n a_i \mathbb{E}[Z_i]$ and $\mathbb{V}[Z] = \sum_{i=1}^n a_i^2 \mathbb{V}[Z_i]$.

2.9.3 Continuous distributions problems

P2.1 Calculate the variance of the uniform distribution $\mathcal{U}(\alpha, \beta)$.

Appendix A

Answers and solutions

Chapter 1 solutions

Answers to exercises

E1.3 4245. E1.13 Mean = 8.9, Min = 5.21, Max = 12.0. E1.14 $Q_1 = 7.76$, Med = 8.69, $Q_3 = 10.35$. E1.15 The frequencies within the bins are 2, 6, 5, and 2. E1.18 $\text{Freq}(\text{debate}) = 8$, $\text{Freq}(\text{lecture}) = 7$, $\text{RelFreq}(\text{debate}) = 0.53$, $\text{RelFreq}(\text{lecture}) = 0.47$. E1.19 The mode is debate with frequency 8.

Solutions to selected exercises

E1.6 Load the data using `pd.read_csv` then call the `.melt()` method.

```
>>> df = pd.read_csv("../datasets/exercises/grades.csv")
>>> df.melt(id_vars=["student_ID"],
            var_name="test",
            value_name="grade")
```

code A.0.1

E1.13 We first use `effort = students["effort"]` to extract the effort variable, then compute the answer using `effort.mean()`, `effort.min()`, and `effort.max()`.

E1.14 Extract the effort variable `effort = students["effort"]` then compute `effort.quantile(q=0.25)`, `effort.median()`, `effort.quantile(q=0.75)`.

E1.15 Run `effort.value_counts(bins=bins).sort_index()` where `bins = [5,7,9,11,13]`.

E1.16 Create the scatter plot by making a data frame then calling `sns.scatterplot`:

```
>>> df = pd.DataFrame([(2,2), (3,3), (4,3), (5,5),
                       (6,4), (5,4), (7,6), (8,5)],
                      columns=["x", "y"])
>>> sns.scatterplot(x="x", y="y", data=df)
```

code A.0.2

E1.17 Use `sns.countplot(x="curriculum", data=students)` to draw a bar chart.

E1.18 Use `students["curriculum"].value_counts()` to obtain the frequencies of the curriculum variable. Add the keyword argument `normalize=True` to obtain the relative frequencies.

E1.19 Call `students["curriculum"].describe()` to get all the info, and observe the top value is debate and its frequency is 8.

Appendix C

Python tutorial

The tutorial is being developed as an interactive notebook. See link below for a preview.

https://nobsstats.com/tutorials/python_tutorial.html